Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

SIMANFOR V2: Desarrollo de una versión mejorada del portal de simulación de gestión forestal SIMANFOR

Rafael Gómez Carrión

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

SIMANFOR V2: Desarrollo de una versión mejorada del portal de simulación de gestión forestal SIMANFOR

Autor: Rafael Gómez Carrión

Tutor: Valentín Cardeñoso Payo

Cotutor: Spyridon Michalakopoulos

La satisfacción radia en el esfuerzo, no en el logro. El esfuerzo total es una victoria completa.
Mahatma Gandhi

Agradecimientos

En primer lugar, agradezco profundamente a mis tutores, quienes han sido una fuente constante de guía, conocimiento y motivación. Su paciencia, dedicación y valiosos consejos han sido esenciales para la culminación de este proyecto. Gracias por creer en mí y por proporcionarme las herramientas necesarias para llevar a cabo este trabajo.

A mi familia, les debo todo lo que soy y todo lo que he logrado. Gracias por su apoyo incondicional, por estar siempre a mi lado y por brindarme el ánimo necesario en los momentos más difíciles. Su apoyo ha sido mi mayor fuente de motivación, y sin ellos este logro no habría sido posible.

A los amigos que he hecho desde que empecé la carrera, les agradezco por estar siempre ahí. En particular, quiero hacer una mención especial a aquellos que comenzaron conmigo pero decidieron seguir otros caminos. Su valentía para tomar decisiones difíciles es inspiradora, y su amistad ha sido invaluable para mí.

A mis amigos fuera de la carrera, gracias por ofrecerme una perspectiva diferente y por las distracciones necesarias que me ayudaron a mantener un equilibrio saludable durante estos años.

Resumen

En este Trabajo de Fin de Grado se ha desarrollado una nueva versión de la plataforma de simulación de entornos forestales SIMANFOR. Esta nueva versión, SIMANFOR V2, tiene como objetivo principal mejorar la usabilidad y la experiencia del usuario durante el uso de la web, centrándose en la conexión directa de los resultados de una simulación con la aplicación RShiny. Para ello, se ha llevado a cabo un proceso de ingeniería inversa sobre la aplicación original, seguido del análisis, diseño, implementación y pruebas de la nueva versión de la plataforma. Como resultado, se ha logrado una aplicación web totalmente funcional que integra de manera efectiva la aplicación RShiny, mejorando significativamente la experiencia del usuario.

Palabras clave: Aplicación web, Ecosistema docker, RShiny, Experiencia de Usuario

Abstract

In this Final Degree Project, a new version of the forest environment simulation platform SIMANFOR has been developed. This new version, SIMANFOR V2, aims to enhance usability and user experience, focusing on the primary objective of directly connecting simulation results with the RShiny application. To achieve this, reverse engineering was performed on the existing application, followed by the analysis, design, implementation, and testing of the new platform version. As a result, a fully functional web application has been obtained, successfully integrating the RShiny application and significantly improving the user experience on the website.

Key words: Web application, Docker ecosystem, RShiny, User experience

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Las herramientas de simulación de entornos forestales constituyen un gran apoyo para la gestión y conservación de este tipo de ecosistemas, permitiendo a los investigadores y gestores modelar y prever la evolución de las masas forestales bajo diferentes escenarios y acciones. Estas simulaciones proporcionan una plataforma segura para probar hipótesis, evaluar políticas de manejo forestal y tomar decisiones sin necesidad de intervenir físicamente en este tipo de entornos, como pueden ser los bosques. Al utilizar estas herramientas, se pueden predecir los efectos de factores como el cambio climático o la explotación forestal, así como de una gran variedad de prácticas de manejo sostenible, contribuyendo a la preservación de los recursos naturales y a la mitigación de impactos ambientales adversos.

Las simulaciones forestales permiten una experimentación controlada y repetible, lo que es crucial para validar modelos y teorías en ecología y gestión forestal. Además, facilitan a los profesionales la divulgación de conocimientos a la comunidad, aumentando la comprensión pública de la importancia de la gestión forestal sostenible. En un contexto donde los recursos naturales están siendo muy controlados debido a la explotación y el cambio climático, las herramientas de simulación se vuelven aún más relevantes. Permiten realizar evaluaciones a largo plazo y ensayar diversas estrategias de manejo, ayudando a identificar las más efectivas para la conservación y el uso sostenible de los bosques [1].

Una de estas herramientas es SIMANFOR [2], una web dedicada a la simulación de entornos forestales. SIMANFOR fue desarrollado por la cátedra de la Universidad de Valladolid, en colaboración con otros centros de investigación y la empresa SNGULAR [3] como encargada de la programación. En ella se permite a los usuarios crear una gran variedad de casos de estudio personalizados, utilizando datos específicos de inventarios forestales, escenarios de manejo y acciones a realizar sobre las masas forestales. A esto se suma la aplicación de modelos empíricos y de procesos para simular la evolución de las masas arbóreas. Los resultados de estas simulaciones se pueden descargar en forma de archivos que pueden ser utilizados para crear gráficos interactivos, mejorando así la interpretación y el análisis de los datos generados. Esta capacidad de simular y visualizar diferentes escenarios es muy importante para la planificación y gestión sostenible de los recursos forestales.

La importancia de herramientas como SIMANFOR radica en su capacidad para integrar datos complejos y proporcionar resultados visuales claros y comprensibles. Esto no solo facilita la toma de decisiones por parte de los gestores forestales, sino que también mejora la comunicación de estos resultados a terceras partes, como pueden ser comunidades locales, responsables políticos o incluso el público en general. La capacidad de simular diversos escenarios y visualizar sus impactos de manera interactiva permite una evaluación más detallada y precisa de las estrategias de manejo forestal.

Por lo tanto, el desarrollo y mejora continua de herramientas como SIMANFOR es vital para mantener su relevancia y utilidad. A medida que se recopilan más datos y se desarrollan nuevos modelos, estas plataformas deben evolucionar para incorporar estos avances, ofreciendo siempre las mejores prácticas y tecnologías disponibles. La integración de nuevas funcionalidades y mejoras en la experiencia de usuario son aspectos cruciales para garantizar que estas herramientas sigan siendo accesibles y útiles para una amplia gama de usuarios.

1.1 Motivación

La principal motivación para llevar a cabo este Trabajo de Fin de Grado surge de la necesidad de mejorar el proceso de visualización de resultados de la web SIMANFOR. Además, al ser una web de reciente creación y en continuo desarrollo, también se considera importante mejorar la experiencia de usuario en la plataforma. Esta iniciativa comenzó con la búsqueda de asesoramiento entre ingenieros informáticos conocidos y, con el apoyo del tutor universitario, se identificó la oportunidad de colaborar con la cátedra de la Universidad de Valladolid dirigida por Felipe Bravo Oviedo. Este grupo de trabajo, al cual pertenece el cotutor del proyecto Spyridon Michalakopoulos, y en colaboración con la empresa SNGULAR, desarrolló la página web de SIMANFOR.

El desarrollo de SIMANFOR representa una innovación significativa en el campo de la gestión forestal. Sin embargo, como ocurre con muchas herramientas tecnológicas, su eficacia puede verse limitada por cuestiones de usabilidad. Entre otros aspectos, la interfaz de usuario juega un papel muy importante en cómo los usuarios interactúan con la herramienta, y en la efectividad con la que pueden utilizar sus funcionalidades. Por ello, mejorar la usabilidad de SIMANFOR no solo facilitará su uso por parte de profesionales, sino que aumentará su capacidad para generar datos precisos y útiles que apoyen decisiones importantes sobre la gestión y conservación de los bosques.

De este modo, se realizó la propuesta para mejorar SIMANFOR al grupo de la cátedra, indicando que el proyecto no modificaría la web original. En su lugar, se propuso desarrollar una nueva versión en paralelo, poniendo el foco en la mejora de la visualización de los resultados de una simulación y añadiendo funcionalidad para mejorar la experiencia de usuario en la web. La cátedra aceptó la propuesta, adoptando así el rol de cliente del proyecto y definiendo los objetivos y necesidades que se querían alcanzar.

Como resultado, se asumió el rol de desarrollador. Se acordó realizar la entrega de la nueva versión de la web a la cátedra tras la finalización del proyecto, para que fuera su equipo el encargado de la integración en la aplicación real. Trabajar en este proyecto proporciona una valiosa oportunidad para aplicar y expandir los conocimientos en ingeniería de software, debido a la variedad de herramientas utilizadas en el desarrollo de la aplicación. Por otro lado, la implementación de mejoras en la plataforma SIMANFOR no solo tiene un impacto directo en la herramienta en sí, sino que también permite adquirir experiencia práctica en un entorno real y colaborativo.

Capítulo 2

Objetivos y Alcance

2.1 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es desarrollar una nueva versión de la plataforma web SIMANFOR, en paralelo a la versión existente, centrándose en mejorar la funcionalidad y usabilidad del sistema. La meta principal es conseguir la integración de la aplicación RShiny en el sistema, facilitando así la visualización gráfica de los resultados de una simulación. Esto permitirá reducir significativamente el número de pasos realizados por los usuarios para ver estos resultados de manera gráfica e interactiva. Este objetivo se considera prioritario debido a su relevancia en la propuesta inicial, ya que fue el que impulsó el desarrollo del proyecto. Además, se definen otros objetivos dirigidos a la mejora de experiencia de usuario en la web.

De este modo, para conseguir desarrollar la nueva versión se definen los siguientes objetivos:

- Facilitar la conexión directa de la pantalla de resultados de una simulación con la aplicación RShiny. Según el diseño de la web inicial, una simulación proporciona un fichero de resultados descargable que debe ser subido manualmente a la aplicación RShiny para visualizar los resultados de forma gráfica. La mejora consiste en reducir significativamente los pasos necesarios para obtener esta representación gráfica.

- Cambiar la ubicación del botón para cerrar sesión, situándolo de forma más intuitiva en la parte superior derecha de la ventana, siguiendo el ejemplo de otras páginas web y facilitando su acceso para el usuario.

- Reorganizar la presentación de los datos mostrados en los listados de usuarios, escenarios, inventarios y modelos. Se revisarán y modificarán dichos datos para incluir únicamente aquellos que sean relevantes para el usuario, basándose en un análisis detallado de los mismos.

- Mejorar la presentación de la documentación de modelos mediante la inclusión de enlaces incrustados en lugar de mostrar simplemente el nombre del archivo. Esta modificación busca hacer la interfaz gráfica más amigable y accesible para el usuario.

- Automatizar el proceso de registro de usuarios. En la versión actual, los nuevos usuarios deben descargar un formulario en formato PDF, completarlo y enviarlo por correo electrónico a SIMANFOR para su validación y creación manual de la cuenta por parte del equipo de desarrollo. El objetivo es crear un formulario dentro de la plataforma que permita a los usuarios introducir sus datos de registro y que el equipo valide o rechace la creación automática del nuevo usuario, sin acudir a sistemas o herramientas externas a la web.

Estas mejoras buscan no solo optimizar la funcionalidad existente y añadir funcionalidad nueva, sino también hacer la plataforma más accesible y fácil de usar para los usuarios, contribuyendo a una experiencia de usuario más fluida y eficiente.

2.1.1 Tareas a realizar

1. Definir los objetivos del proyecto para elaborar una planificación detallada.
2. Estudiar la estructura de la web existente, realizando ingeniería inversa sobre los repositorios en Github.
3. Poner en funcionamiento la máquina virtual sobre la que se desarrolla el proyecto.
4. Configurar el entorno de desarrollo para trabajar con todo el sistema en local.
5. Implementar los objetivos propuestos.
6. Realizar pruebas sobre las soluciones implementadas.
7. Documentar cualquier detalle adicional y finalizar la memoria del proyecto.
8. Proporcionar el código de la nueva versión a SIMANFOR para su integración en la web real.

2.2 Alcance

Tal y como se ha explicado anteriormente, el objetivo principal de este proyecto es mejorar la funcionalidad y la experiencia de usuario de la web SIMANFOR. Aunque el foco principal estará en el frontend de la aplicación, también se pueden requerir ajustes en algunos elementos del *backend* o en la estructura Docker sobre la cual se sustenta la aplicación.

Por otro lado, es necesario establecer los límites del proyecto, definiendo las tareas o funcionalidades que quedan fuera de su alcance y que por tanto no serán abordadas:

- No se realizarán modificaciones en las funcionalidades relacionadas con las simulaciones, con el objetivo de mantener la función principal del sitio sin alteraciones.
- Se evitarán cambios en cualquier elemento o concepto vinculado directamente con la ingeniería forestal, ya que estos aspectos están fuera del ámbito de la ingeniería informática.
- La funcionalidad para el simulador de Smartelo, enfocándose exclusivamente en el simulador SIMANFOR.

- No se llevarán a cabo modificaciones en la web original. El proyecto consistirá en desarrollar una nueva versión de la web que funcionará de manera independiente, la cual será entregada al equipo de desarrollo de SIMANFOR al finalizar el proyecto.
- En relación con el punto anterior, no integrarán las funcionalidades de la nueva versión en la web original. Esta tarea corresponderá al equipo de desarrollo de SIMANFOR, tal como se acordó al inicio del proyecto.

Este enfoque asegura que el proyecto se mantenga dentro de un alcance manejable y enfocado, permitiendo aplicar y demostrar habilidades en ingeniería de software mientras se mejora significativamente una herramienta importante en la gestión forestal.

Capítulo 3

Metodología

En primer lugar, es necesario detallar la forma en la que se identificaron los objetivos del proyecto. Se llevaron a cabo una serie de entrevistas y reuniones con el director del equipo de desarrollo de SIMANFOR, en presencia también de los tutores del proyecto. Durante estas reuniones, se definieron claramente los objetivos y tareas a realizar, tras lo cual se realizó una presentación del proyecto ante el equipo completo de SIMANFOR, el 26 de octubre de 2023.

El siguiente paso corresponde a la planificación del proyecto. Se estableció un horario de trabajo de lunes a viernes de 10:00h a 14:00h. Además, se decidió dividir el proyecto en tres fases, las cuales se van a citar aunque en el siguiente apartado se detallarán más específicamente:

- La primera fase consistió en el estudio de la estructura de la web a partir de los repositorios almacenados en Github, realizada durante noviembre de 2023.
- La segunda fase implicó la configuración del entorno de trabajo en local, lo que incluyó la configuración de una máquina virtual. Esta fase experimentó retrasos debido a problemas técnicos, prolongándose durante diciembre de 2023 y enero de 2024.
- La tercera fase se centró en el desarrollo e implementación de los objetivos y tareas propuestos, alternando con la redacción de la memoria del proyecto. Esta fase se extendió desde febrero hasta mayo de 2024.

En cuanto a la metodología de la implementación, se optó por un modelo de desarrollo incremental debido a la naturaleza del proyecto, que se centra en la consecución de objetivos. El modelo incremental parte de la premisa de proporcionar un crecimiento progresivo de la funcionalidad. Así, en este proyecto cada objetivo identificado representa un incremento en el desarrollo, proporcionando una versión completa del sistema tras la consecución de cada uno de ellos [4].

En el modelo de desarrollo incremental es necesario establecer una prioridad sobre los objetivos a conseguir. Esta priorización se basó en las directrices proporcionadas por el cliente, SIMANFOR. Se comenzó por los objetivos de mayor prioridad, desarrollando su funcionalidad y presentando la versión final al cliente antes

de avanzar a la siguiente tarea. Esto permitió incorporar el feedback del cliente y realizar ajustes según fuera necesario. Al seguir esta metodología, se logró un enfoque estructurado y eficiente para la consecución de los objetivos del proyecto, garantizando la satisfacción del cliente y la entrega exitosa de los resultados esperados.

3.1 Fases y costes

3.1.1 Fases del proyecto

En este apartado se detallarán las fases planificadas para el desarrollo del proyecto antes de su inicio. Durante el desarrollo, estas fases sufrieron algunos cambios debido a retrasos provocados, principalmente, por problemas técnicos con el entorno de trabajo. Una vez presentadas las fases propuestas al inicio y los retrasos que se produjeron, se incluirá un diagrama de Gantt que mostrará específicamente la duración real de cada una de ellas.

La primera fase del proyecto consiste en el análisis de la aplicación web SIMANFOR desde la perspectiva de un usuario. El objetivo de este análisis es definir los requisitos y funcionalidades que se añadirán a la web, las cuales serán el foco del proyecto. Para ello, se cuenta con la colaboración del equipo de SIMANFOR, con el fin de definir conjuntamente los objetivos de mejora en la interfaz de usuario. Esta tarea se estimó en dos semanas, tras las cuales se realiza la presentación de los objetivos del proyecto al equipo de SIMANFOR.

En la segunda fase se especifican las tareas e incrementos que componen el proyecto. Se definen las tareas previas a la implementación, así como los incrementos del proceso de desarrollo, asegurando que cada versión de la aplicación web sea totalmente funcional. Esta fase tiene una duración estimada de una semana.

La siguiente fase abarca el análisis de los repositorios de Github. Se planea un proceso de ingeniería inversa para documentar todo el contenido de los tres repositorios proporcionados por SIMANFOR: *production*, *web-backend*, y *web-frontend*. Esta tarea tiene una duración estimada de un mes (cuatro semanas), tiempo suficiente para realizar un estudio exhaustivo.

Después, se define la fase de configuración del entorno de trabajo para el desarrollo del proyecto. Consiste en configurar una máquina virtual proporcionada por el tutor del proyecto y clonar a esta máquina los repositorios Github de SIMANFOR. Además, se debe configurar el ecosistema Docker para que opere localmente, sin afectar al entorno de producción. Originalmente planificada para durar un mes, esta fase experimentó retrasos que extendieron su desarrollo a dos meses (ocho semanas).

Una vez definidas las tareas previas al desarrollo del proyecto, se define la fase de implementación del proyecto; esta se divide, a su vez, en diferentes iteraciones del proceso de desarrollo incremental, alineadas con los objetivos iniciales de mejora de la experiencia de usuario en la web. La primera iteración consiste en conectar la web SIMANFOR con la aplicación RShiny, permitiendo obtener resultados gráficos de una simulación sin salir del entorno web. Aunque se planificaron seis semanas para esta tarea, su desarrollo se extendió a diez semanas debido a problemas técnicos.

Debido a los retrasos en las fases anteriores, las siguientes iteraciones del desarrollo incremental vieron reducido su tiempo respecto al planificado. Estas iteraciones comprendían los siguientes objetivos: modificación de la ubicación del botón *Cerrar Sesión*; reorganización de los datos en tablas de usuarios, inventarios, modelos

y escenarios; visualización del enlace a la documentación de los modelos; y automatización del proceso de creación de nuevos usuarios. Inicialmente planificadas para ocho semanas, estas tareas se completaron en cinco semanas.

La última iteración del desarrollo incremental es la fase de pruebas del proyecto. Durante este periodo se realizan pruebas para verificar la funcionalidad añadida a la web, comprobando también las restricciones en las operaciones y la robustez de la nueva versión de la aplicación. Esta fase tiene una duración planificada de una semana.

Por último, se detalla la fase de escritura de la memoria del Trabajo de Fin de Grado. En este período se realizará tanto esta escritura como su validación para posterior entrega, estimando como duración tres semanas para el proceso.

En la Figura 3.1 se presenta el diagrama de Gantt con la duración real de las fases detalladas. Este diagrama refleja los tiempos reales tras los retrasos experimentados en algunas fases. El inicio del proyecto tuvo lugar la semana del 16 de octubre de 2023 y finalizó la semana del 3 de junio de 2024 con la escritura de la memoria.

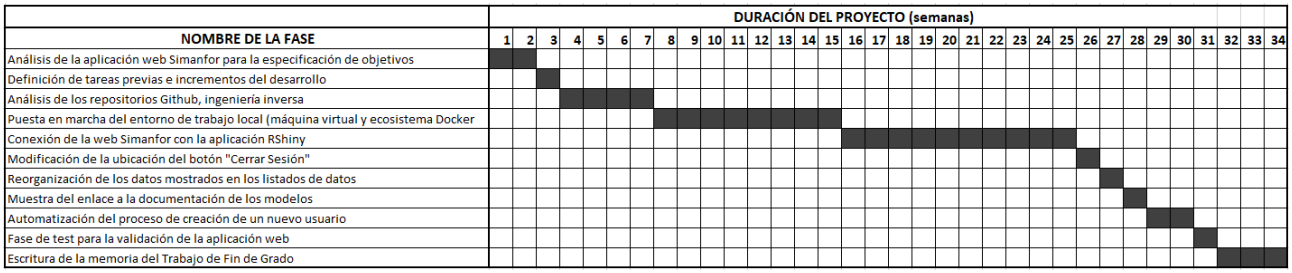

Figura 3.1: Diagrama de Gantt donde se detallan las fases del proyecto

3.1.2 Costes del proyecto

Para detallar los costes del proyecto, se distinguirán entre componentes hardware y software. En ellos se especificarán las herramientas, programas y otros elementos utilizados para el desarrollo de la aplicación web.

Componentes Software

Dado el carácter académico del proyecto (un Trabajo de Fin de Grado universitario), se emplean licencias de la Universidad de Valladolid para Office 365 y Astah Professional. El resto de las herramientas no suponen ningún coste, ya que son de código abierto (*open source*) o proporcionadas por SIMANFOR. En el capítulo 5 se especifican todos los programas y herramientas software utilizadas durante el desarrollo del proyecto.

Componentes Hardware

En la Tabla 3.1 se detallan los costes de los elementos hardware utilizados en el desarrollo del proyecto. Dado que no se han asumido costes de software, la totalidad del coste del proyecto se refleja en este apartado. Estos costes comprenden tanto el equipo utilizado para el desarrollo del proyecto como los servidores y máquinas

virtuales a los que se accede para realizar el trabajo. En el caso de estos últimos, se utilizan una máquina virtual proporcionada por el tutor de la UVa y el servidor de simulaciones proporcionado por SIMANFOR, por lo que no se añade ningún coste adicional.

Herramientas Hardware	**Precio (€)**
Ordenador Portátil	600
Monitor adicional	200
Teclado adicional	100
Ratón inalámbrico	50
Servidor de la máquina virtual	0
Servidor de simulaciones SIMANFOR	0
Conexión a Internet	50/mes

Tabla 3.1: Costes de componentes software

Capítulo 4

Aplicación Existente

Este proyecto se basa en el análisis y desarrollo de una aplicación web preexistente, SIMANFOR, que ya está en uso en su entorno de producción. La empresa proporcionó los repositorios de GitHub que contienen todos los archivos necesarios para desplegar la infraestructura de la aplicación, basada en contenedores Docker para facilitar una infraestructura modular y escalable. Por tanto, la primera etapa del trabajo consistió en un proceso de ingeniería inversa, analizando exhaustivamente la estructura y funcionalidad de estos archivos para comprender la arquitectura existente y cómo se lleva a cabo el despliegue de la aplicación.

4.1 Arquitectura de la aplicación

La aplicación SIMANFOR está diseñada para facilitar la simulación forestal, proporcionando herramientas robustas para el análisis y visualización de datos forestales. La arquitectura de la aplicación está compuesta por varios componentes clave, cada uno con funciones específicas que contribuyen al funcionamiento global del sistema. Estos componentes son gestionados y desplegados utilizando Docker, lo que permite una fácil administración y escalabilidad. A continuación se describen estos componentes y su rol en la arquitectura de la aplicación:

- Base de datos (MongoDB): Almacena todos los datos relevantes de la aplicación, incluyendo usuarios, acciones, inventarios, modelos y escenarios. MongoDB se elige por su flexibilidad y capacidad para manejar grandes volúmenes de datos no estructurados.

- *backend* (Node.js + Express): Gestiona la lógica de la aplicación, incluyendo la comunicación con la base de datos y la ejecución de los scripts necesarios para iniciar nuevas simulaciones. También expone una API para realizar la comunicación con el frontend.

- Frontend (Angular 9): Proporciona la interfaz de usuario a través de la cual los usuarios interactúan con la aplicación. Este componente se comunica con la API del *backend* para mostrar datos y resultados.

- Proxy inverso (Nginx): Maneja las conexiones web, dirigiendo las peticiones del usuario al frontend o al *backend* según corresponda. También gestiona la seguridad de las conexiones mediante la implementación de HTTPS y otras medidas de seguridad.
- Scripts de simulación: Ejecutan nuevas simulaciones en un servidor separado, comunicándose con el *backend* a través de SSH.

4.2 Despliegue de la aplicación

El despliegue de SIMANFOR se gestiona mediante Docker, lo que permite una administración fácil y una escalabilidad adecuada para la aplicación. Para ello se utilizan archivos de configuración que especifican cómo deben ser creados y ejecutados los contenedores. Estos archivos se encuentran en el repositorio *production*; a continuación, se describe su contenido y su funcionalidad:

- *main.yml*: en él se especifica que los contenedores Docker utilizados en el despliegue de la aplicación son alojados y gestionados por el servicio GHCR de GitHub. Este servicio permite almacenar imágenes y asociarlas con repositorios completos de GitHub, además de permitir el uso de imágenes públicas de manera anónima [5]. En este fichero también se describe el proceso de creación y despliegue de la estructura Docker, el cual involucra una serie de pasos que culminan con la eliminación de la estructura previa (si existe) para asegurar la operatividad de una única estructura Docker.
- *docker-compose.yml*: es el fichero más importante para la organización del despliegue de Docker, ya que en él se definen los servicios y contenedores necesarios para que la aplicación funcione correctamente [6]. En él se coordina la creación y ejecución de los siguientes contenedores:
  - *mongo*: Contiene una instancia de MongoDB, la base de datos utilizada en la aplicación. Para su creación se utiliza una imagen oficial de MongoDB, además de un volumen configurado para el almacenamiento persistente de los datos. En este caso, las tablas almacenadas en la base de datos incluyen usuarios, acciones, inventarios, modelos y escenarios.
  - *backend*: Construido a partir de una imagen definida en su *Dockerfile*, en él se definen variables de entorno para conseguir la comunicación con MongoDB y con el servidor de simulación. También se definen volúmenes específicos que facilitan la gestión de datos de entrada y salida de las simulaciones [7].
  - *frontend*: Este contenedor se construye y despliega después del *backend*, utilizando también una imagen definida en su *Dockerfile*. En él se asegura que la interfaz de usuario esté disponible para interactuar con la lógica y datos gestionados por el *backend*.
  - *proxy*: Se basa en una imagen predeterminada de Nginx [8], configurada para manejar las conexiones web y dirigirlas al frontend o al *backend* según la URL de las peticiones, utilizando para ello el fichero *nginx.conf*.
- *nginx.conf*: es el encargado de la configuración del proxy inverso. Este tipo de software está diseñado para aplicaciones web con alta concurrencia, buen rendimiento y bajo uso de memoria, configurándolo de manera que se asegura que el tráfico se dirige adecuadamente, mejorando la eficiencia y la seguridad de la web [9]. En él se detalla como puerto de entrada del contenedor el 443, diferenciando entre las peticiones dirigidas al frontend (indicando la URL /) y al *backend* (URL /api/).
- *Makefile*: facilita la ejecución simultánea de comandos para la gestión del repositorio *production* y los contenedores Docker. Se definen varias acciones: help, clone, update, up, stop y clean, cada una destinada a simplificar tareas administrativas y de mantenimiento.

En resumen, el repositorio *production* contiene los archivos que definen las configuraciones necesarias para desplegar y mantener la aplicación SIMANFOR en un entorno de producción. Para ello se utilizan contenedores Docker que aseguran una infraestructura modular, escalable y fácil de gestionar.

4.3 Estructura del *backend*

El *backend* de la plataforma SIMANFOR es una API REST desarrollada en Node.js, que actúa como intermediario entre la base de datos MongoDB, el servidor de simulación de SIMANFOR (la comunicación con este servidor se establece mediante el protocolo SSH) y la interfaz de usuario del frontend. Esta sección describe los componentes y funcionalidades clave del *backend*, así como su despliegue mediante Docker.

4.3.1 Arquitectura del *backend*

El *backend* de SIMANFOR está implementado en Node.js y utiliza el framework Express para manejar las solicitudes HTTP. La API REST expone varios endpoints que permiten la gestión de los usuarios, inventarios, modelos y escenarios de simulación. La documentación de la API se genera y mantiene utilizando Swagger (OpenAPI) [10], facilitando su uso y comprensión por parte de desarrolladores y sistemas externos. Su mayor utilidad radica en la capacidad de generar su propia documentación, de forma que sea legible por otras máquinas a partir de una descripción detallada contenida en un archivo YAML o JSON [11].

La especificación de la API se encuentra en el archivo *openapi.yaml*, ubicado en el directorio *docs*. Este archivo define la versión de OpenAPI (3.0.0), el título del proyecto (Simanfor), la versión de la API (2.0.2) y los servidores disponibles (un servidor local y el servidor de despliegue). También describe las rutas y operaciones disponibles en la API, así como los tipos de datos utilizados y el esquema de seguridad basado en una clave de API (apiKey) incluida en la cabecera de las solicitudes.

4.3.2 Despliegue del *backend*

El despliegue del *backend* se gestiona mediante Docker, utilizando un *Dockerfile* para definir la imagen del contenedor. La imagen se basa en una versión predeterminada de Node.js [12], cuyo objetivo principal es crear aplicaciones orientadas a maximizar el rendimiento y la eficiencia, con ejecución en tiempo real por parte del usuario. La imagen se construye a través de los siguientes pasos:

1. Creación del directorio de trabajo: Se establece un directorio donde se almacenarán los archivos necesarios para la ejecución.

2. Instalación de dependencias: Las dependencias de la aplicación se instalan utilizando los archivos *package.json* y *package-lock.json*.

3. Ejecución de Node.js: El servidor de Node.js se configura para escuchar en el puerto 3000, tras lo cual se ejecuta el archivo *index.js* como punto de partida del despliegue de la API.

4.3.3 Funcionalidades del *backend*

El archivo *index.js* configura el entorno de ejecución de la aplicación, definiendo el puerto de escucha del *backend* y cargando una serie de módulos esenciales para el desarrollo web: *body-parser*, *express-fileupload*, *yamljs*, *swagger-ui-express*, *cors* y *express*. Estos módulos facilitan la gestión de solicitudes HTTP, la manipulación de archivos, la carga de archivos YAML, la integración con Swagger, la comunicación con servidores externos y el desarrollo general utilizando el framework Express.js.

Los demás módulos cargados en el fichero *index.js* corresponden con otros archivos de la aplicación, cada uno responsable de diferentes aspectos de la misma:

- *common.js*: Proporciona funcionalidad básica a través de cuatro funciones esenciales que se exportan al cargar el módulo. Estas funciones incluyen la gestión de errores, la confirmación de operaciones exitosas, la actualización de archivos y la validación de argumentos, siendo utilizadas en la mayoría del resto de componentes del *backend*.
- *inventory-check.js*: Valida que los datos proporcionados en un inventario sean correctos. Se validan archivos en formatos XLSX y CSV.
- *mongo.js*: Es uno de los componentes más importantes, ya que gestiona la comunicación con MongoDB. Su función inicial es establecer la conexión para ejecutar todas las queries pendientes, para posteriormente definir las cuatro funciones básicas de comunicación con la base de datos: inserción, búsqueda, actualización y eliminación de registros.
- *ssh.js*: Maneja la conexión SSH con el servidor de simulación. Contiene funciones que permiten ejecutar comandos en el servidor, monitorear los trabajos que se están realizando en él, mandar un escenario para iniciar su simulación y transferir otro tipo de archivos mediante el protocolo SCP.
- *smartelo.js*: Incluye funcionalidades relacionadas con el simulador Smartelo. Como este proyecto se centra exclusivamente en SIMANFOR, la funcionalidad de esta herramienta no será analizada.

El resto de archivos definen las funciones disponibles en la API, indicando sus rutas y métodos disponibles:

- *auth.js*: Administra la funcionalidad relacionada con los usuarios. En él se gestiona la autenticación y autorización de usuarios, incluyendo métodos para el login, registro de un nuevo usuario, cambio de contraseña y eliminación de usuarios.
- *monitorize.js*: Se encarga de gestionar el estado de las simulaciones del servidor de simulación. Para ello, obtiene un listado de los trabajos que están en cola o siendo simulados, para después cambiar su estado en caso de ser necesario (a *RUNNING* o *FINISHED*), y descargar sus resultados cuando ya ha finalizado su simulación.
- *inventory.js*: Maneja las operaciones relacionadas con los inventarios. Estas incluyen obtención de todos los inventarios de la base de datos, obtención de uno específico, descarga del archivo de datos, subida, actualización y eliminación de la base de datos de un inventario.
- *model.js*: Similar al anterior, gestiona los métodos relacionados con los modelos en la base de datos. Permite obtener todos los modelos, obtener uno específico, añadir, modificar o eliminar un modelo de la base de datos.

- *scenario.js*: A diferencia de los anteriores, en este se incluye una función para almacenar los pasos de un escenario en un objeto JSON, utilizando como plantilla el archivo *scenario-template.json*. Este es el archivo que se guarda en el directorio *input/scenario* cuando se sube un escenario a la base de datos. Además de esta operación, se incluyen métodos para obtener todos los escenarios, obtener solamente uno, poner en marcha una simulación, descargar el archivo con sus resultados y eliminar un escenario existente.

4.4 Estructura del frontend

El frontend de la plataforma SIMANFOR está desarrollado sobre Angular 9 [13], un framework de código abierto que facilita la creación de aplicaciones web de una sola página (SPA, Single Page Application) donde las pantallas son vistas que se van intercambiando. Angular es conocido por su robustez y modularidad, utilizando TypeScript y JavaScript, junto con HTML y CSS, para construir interfaces de usuario interactivas y dinámicas. A continuación, se detalla la estructura del frontend, dividiéndola en su arquitectura, despliegue y funcionalidades clave.

4.4.1 Arquitectura del frontend

La arquitectura del frontend en SIMANFOR está basada en la modularidad de Angular, permitiendo una organización clara y escalable del código. Los elementos por los que está formada esta estructura son:

- Estructura de Módulos: El proyecto Angular utiliza varios módulos que encapsulan diferentes funcionalidades. El módulo principal es *AppModule*, definido en el archivo *app.module.ts*. En él se especifican los componentes que pertenecen a este módulo, los módulos restantes importados, los servicios utilizados (en este caso se define *ApiService*) y el componente raíz (el componente *app*).
- Componentes: Los componentes son los bloques de construcción de la interfaz de usuario. Cada componente tiene un archivo HTML para la estructura, un archivo CSS/SCSS para los estilos, un archivo TypeScript para la lógica, y un archivo de pruebas (spec.ts) para pruebas unitarias.
- Servicios: Los servicios en Angular se utilizan para manejar la lógica de negocio y la comunicación con el *backend*, facilitando la separación de preocupaciones y promoviendo la reutilización del código.
- Modelos de datos: Definen los tipos de objetos utilizados en los componentes y servicios, al recibir esos objetos como resultados de consultas lanzadas al *backend* de la aplicación.
- Enrutamiento: Angular Router se utiliza para gestionar la navegación entre diferentes vistas de la aplicación. El archivo *app.routing.ts* define las rutas principales utilizando un array de objetos de ruta, e indica el componente *FullComponent* como principal y *HomeComponent* como secundario.
- Herramientas de Pruebas: Los archivos *karma.conf.js* y *protractor.conf.js* configuran las herramientas Karma [14] y Protractor [15], respectivamente. Karma se utiliza para ejecutar pruebas unitarias sobre JavaScript en diferentes navegadores, definiendo los framework de pruebas, los complementos necesarios, los reporters y los navegadores, entre otros elementos. Protractor automatiza las pruebas extremo a extremo (E2E), asegurando que la aplicación funciona correctamente desde la perspectiva del usuario, a través de la definición del tiempo de espera, las especificaciones de las pruebas, las capacidades del navegador, la URL base y el framework de pruebas, entre otros.

- Archivos de configuración: Se trata de ficheros que definen tanto características generales de la aplicación Angular como ciertas herramientas utilizadas para dar soporte a funcionalidades concretas.

Entre los archivos de configuración más importantes contenidos en el frontend se encuentran:

- *.editorconfig*: Mantiene la coherencia en el estilo de codificación entre diferentes editores y entornos de desarrollo.
- *polyfills.ts*: Asegura la compatibilidad con diferentes navegadores, especialmente en aquellos que no admiten ciertas características de JavaScript de manera nativa.
- *tsconfig.json*, *tsconfig.app.json* y *tsconfig.spec.json*: Agregan características al compilador de TypeScript, específicamente para la aplicación y para archivos de prueba. Por ejemplo, el segundo incluye al archivo *main.ts* como punto de entrada de la aplicación [16].
- *test.js*: Configura el entorno de pruebas de Angular, cargando dinámicamente los archivos de especificación de pruebas e iniciando Karma para ejecutarlas.
- *angular.json*: Es el archivo de configuración principal del proyecto Angular, definiendo la configuración global de Angular CLI y las específicas para cada proyecto dentro de la aplicación. En este caso, se definen dos proyectos: *material* y *material-e2e*. El primero define el proyecto principal, siendo definido como una aplicación y especificando *src* como su directorio fuente; el segundo especifica la configuración para las pruebas extremo a extremo del proyecto [17].

4.4.2 Despliegue del frontend

Al igual que el *backend*, el despliegue del frontend se realiza utilizando Docker y GitHub Actions para la automatización. La configuración del despliegue incluye los siguientes elementos:

- Archivo *Dockerfile*: Define el proceso de construcción de la imagen Docker que utiliza el contenedor donde se aloja la aplicación del frontend. Este proceso se divide en dos etapas, *Build* y *Run*. La primera utiliza Node.js para construir la aplicación, copiando los archivos de configuración del entorno (*package.json*, donde se definen metadatos del proyecto, como su dependencias y scripts de construcción, y *package-lock.json*, donde se detallan las versiones exactas de las dependencias instaladas), para después lanzar el comando que inicia la construcción. La segunda utiliza Nginx (con su configuración definida en el archivo *nginx.conf*) para servir el contenido estático, definiendo la forma en que se tratan las peticiones dirigidas al frontend.
- GitHub Actions: El archivo *main.yml* del directorio *.github/workflows* automatiza la construcción y despliegue de la imagen Docker. Los pasos incluyen comprobación del repositorio Github, autenticación en el registro de contenedores, extracción de metadatos Docker, construcción, etiquetado y subida de la imagen, y limpieza de versiones anteriores.
- Firebase Hosting: Es el servicio utilizado para alojar la aplicación web (es decir, el hosting de la aplicación), configurado a través del archivo *firebase.js*. En él se especifica la carpeta pública, los archivos a ignorar y las reglas de reescritura de URL, donde se define que todas las rutas se redirigen a la página principal (*index.html*), para que la aplicación SPA pueda manejarlas correctamente [18].

Otro elemento importante para el despliegue de la aplicación es el contenido del directorio *environments*: en él se define cómo el frontend de Angular se conecta con la API del *backend*. Para ello se definen dos archivos, donde se definen un entorno local (se accede al *backend* a través de *http://localhost:3000*) y uno de producción (se accede al *backend* a través de *https://www.simanfor.es/api*).

4.4.3 Funcionalidades del frontend

En el frontend de SIMANFOR, los componentes y servicios están diseñados para ofrecer una experiencia de usuario fluida e intuitiva, facilitando el acceso y la gestión de datos forestales. A continuación, se describe la funcionalidad de los principales elementos del frontend.

En primer lugar, se indica el contenido de los directorios *pipes* y *assets*, utilizados en el resto de componentes de la aplicación. El primero contiene un archivo pipe personalizado utilizado para recortar texto, mientras que el segundo almacena documentos (en general son manuales de uso, formularios e información legal), archivos de idiomas (los idiomas disponibles son español, inglés, francés, euskera, gallego, portugués y vietnamita), imágenes y archivos de estilos (se define el estilo de las páginas, cabeceras y barra lateral, entre otros elementos).

Por otro lado, como se define en los archivos de configuración y despliegue del frontend, existen dos archivos que funcionan como punto de entrada de la aplicación:

- *main.ts*: Realiza dos funciones principales: por un lado, habilita el modo de producción de Angular en caso de que la aplicación lo esté para optimizar rendimiento; por otro lado, inicializa la aplicación en el entorno del navegador, indicando su módulo raíz (*AppModule*).
- *index.html*: Proporciona la estructura básica y los metadatos necesarios para cargar y representar la aplicación Angular correctamente en el navegador. También define la etiqueta a partir de la cual se monta dinámicamente y se controla todo el contenido de las páginas de la aplicación (el componente *app*).

Componentes del frontend

El componente *app* es el componente de entrada, funciona como la raíz de la aplicación. En él se define el contenedor principal de las páginas de la web, donde se carga dinámicamente su contenido según la ruta de la URL. Además, gestiona los consentimientos de cookies y el servicio de traducciones, configurando el español como idioma predeterminado, y la eliminación de suscripciones a eventos cuando se destruye el componente, para evitar problemas en memoria.

En el directorio *layouts/full* se incluyen los componentes que definen el diseño global de las páginas de la aplicación, con elementos comunes en todas ellas. Estos componentes son:

- Componente *full*: Es el principal, ya que define el encabezado, la barra lateral y el contenido principal de la página. Utiliza Angular Router par decidir qué componente se carga en ella.
- Componente *header*: Se sitúa en el encabezado de las páginas, y muestra el selector de idioma y un indicador de carga.

- Componente *sidebar*: Muestra el menú lateral con los diferentes elementos de la web, así como las opciones de inicio y cierre de sesión del usuario.

Se definen una serie de componentes de utilidades, contenidos en el directorio *pages/utils*. Estos proporcionan funcionalidades comunes utilizadas en varias partes la aplicación:

- Componente *table*: Establece una estructura de tabla general para presentar listados de elementos en una página de la web, mostrando la información específica requerida en ella. Se define el título de la tabla, un botón para añadir nuevos elementos y la estructura de la tabla en sí. Los datos de la tabla son dinámicos, donde cada fila representa un registro de la base de datos, y en cada celda se muestra el dato cuyo nombre coincide con el encabezado de la columna. Además, se añaden botones de acción para cada fila, mostrados en la última celda de cada una. En cuanto a la lógica, se define una plantilla que se agrega a los archivos HTML de los componentes que necesitan una tabla, donde se definen los eventos que se pueden desencadenar desde los botones de acción. Se definen métodos para manejar la selección de filas, mostrar un cierto número de elementos en la tabla, generar elementos para mostrar en una columna según su encabezado, manejar las acciones según el botón de acción pulsado y agregar un nuevo elemento.

- Componente *confirmation*: Proporciona un mensaje de confirmación y dos botones: uno para aceptar dicho mensaje y otro para cancelarlo. Su función es proporcionar al usuario la posibilidad de cancelar ciertas acciones (por ejemplo, al pulsar el botón que elimina un inventario de la base de datos).

- Componente *addElement* (figura 4.1): Presenta un formulario que permite añadir un nuevo elemento a la base de datos. En ese caso el formulario se muestra vacío, pero también puede mostrar los datos de un elemento si se selecciona la opción de editarlo en su listado correspondiente. Cuando se cierra, se devuelve un objeto con los datos del elemento que se debe añadir o actualizar en la base de datos.

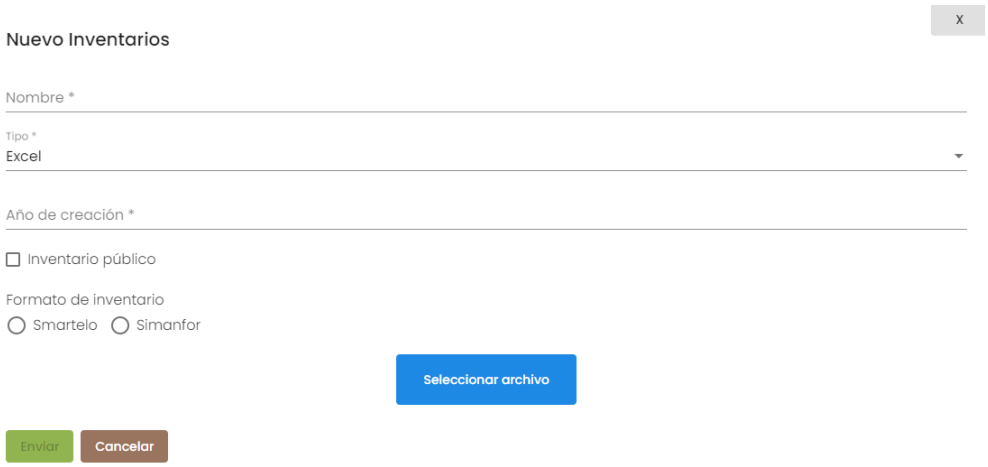

Figura 4.1: Componente *addElement* de la web SIMANFOR

- Componente *addScenario* (figura 4.2): Muestra el formulario donde se definen los elementos de un escenario. En la parte superior se muestran botones para seleccionar el inventario y el modelo de proyección asociados al escenario. En la parte central se definen los pasos del escenario, incluyendo botones para añadir o eliminar un paso. Por último, en la parte inferior se definen tres botones: uno para cancelar y cerrar el componente, otro para subir el escenario creado a la base de datos y otro para comenzar una simulación, mostrando este último solamente cuando se selecciona la opción de editar un escenario existente.

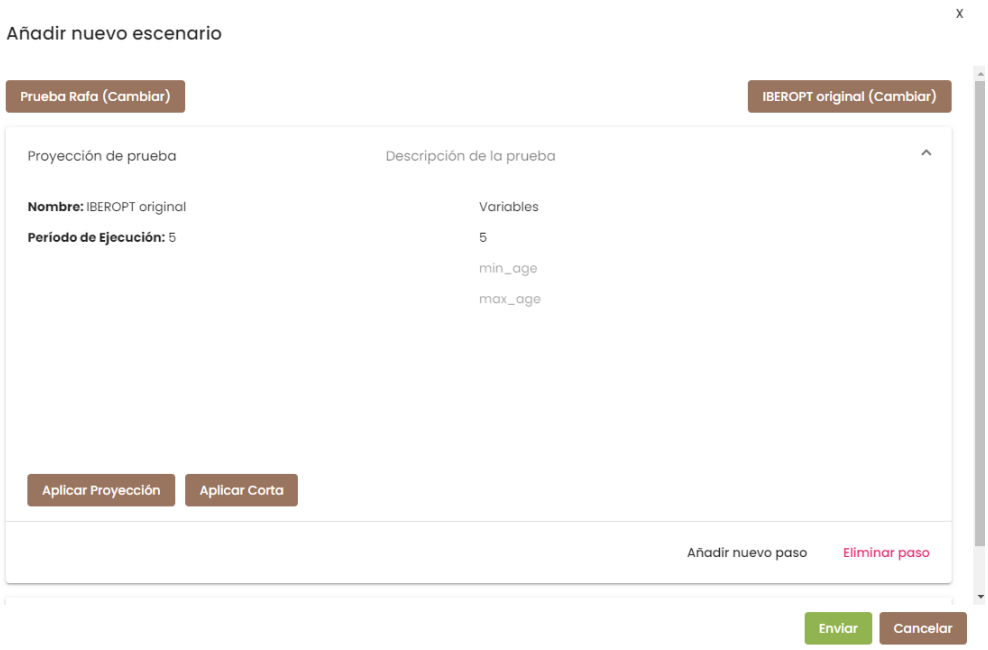

Figura 4.2: Componente *addScenario* de la web SIMANFOR

Los componentes del directorio *pages* representan las páginas específicas de la aplicación. Además, en él se incluyen dos archivos importantes, *pages.module.ts* y *pages.routing.ts*. El primero organiza los elementos relacionados con las páginas y vistas de la aplicación, y el segundo define las rutas para las páginas y especifica a cuáles puede acceder un usuario autenticado, utilizando el servicio *authGuardService* para ello. Los componentes de este directorio son:

- Componente *home* (figura 4.3): Es la primera página a la que accede el usuario al conectarse a la web. En ella se presenta información sobre SIMANFOR, así como un enlace para descargar el documento de solicitud de nueva cuenta de usuario. En su lógica se comprueba si el usuario ya está autenticado, para en ese caso navegar a la página *inventory* de la web.

Figura 4.3: Página *home* de la web SIMANFOR

- Componente *cookies*: Es una página vacía, cuyo contenido podría tener que ver con información acerca de la política de cookies utilizada en la web.

- Componente *help*: Proporciona funcionalidad visual, presentando recursos de ayuda y enlaces útiles para el uso de la web.
- Componente *legal*: Similar a la anterior, muestra los términos y condiciones de uso de SIMANFOR.
- Componente *login* (figura 4.4): Define la página de inicio de sesión. En ella se muestra un formulario para escribir nombre de usuario y contraseña, así como enlaces a las condiciones de uso y al documento con el formulario de registro. Su lógica define que si el usuario está autenticado se redirige a la página *inventory*, así como el método que se ejecuta cuando se pulsa el botón de confirmación de inicio de sesión. En él, se crea un objeto de tipo FormData que se envía a través de una solicitud HTTP POST al *backend*, y se recibe como respuesta un token y un rol, los cuales se utilizan para autenticar al usuario durante su estancia en la web.

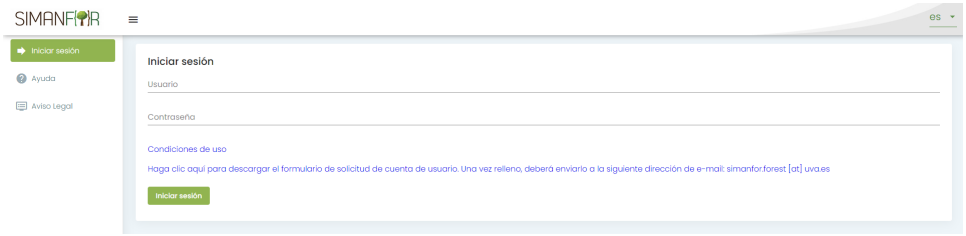

Figura 4.4: Página *login* de la web SIMANFOR

- Componente *users* (figura 4.5; solamente se muestra un usuario ejemplo, tapando los datos del resto por cuestiones de privacidad): Esta página solo es accesible para usuarios administradores, ya que contiene un listado que muestra todos los usuarios registrados en la plataforma. Su archivo HTML muestra únicamente una instancia del componente *table*, que define la tabla donde se muestra el listado. En ella se rellenan primero las cabeceras de las columnas, con la información de los usuarios a mostrar. Después, se rellenan las filas correspondientes a cada usuario de la base de datos. Además, se definen dos métodos, el primero asociado al botón de añadir un nuevo usuario, que abre el componente *addElement*, y el segundo a los botones de cada fila de la tabla, que permiten eliminar de la base de datos el usuario de dicha fila.

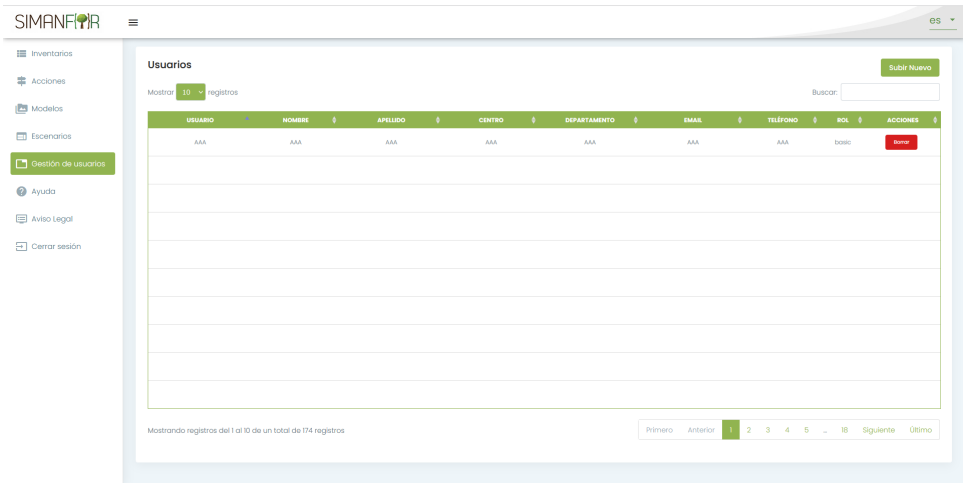

Figura 4.5: Página *users* de la web SIMANFOR

- Componente *inventory* (figura 4.6): Es la página donde se muestra el listado de los inventarios accesibles por un usuario, dependiendo del rol que tengan en la web. Los administradores ven todos los inventarios, mientras que los usuarios básicos solo ven los inventarios públicos y los que han sido subidos por ellos.

Su contenido es igual que el del componente *users*, una instancia de *table* con sus datos mostrados según corresponda. Se añade un botón adicional a cada fila de la tabla, el cual permite editar los datos del inventario correspondiente mediante la muestra del componente *addElement*.

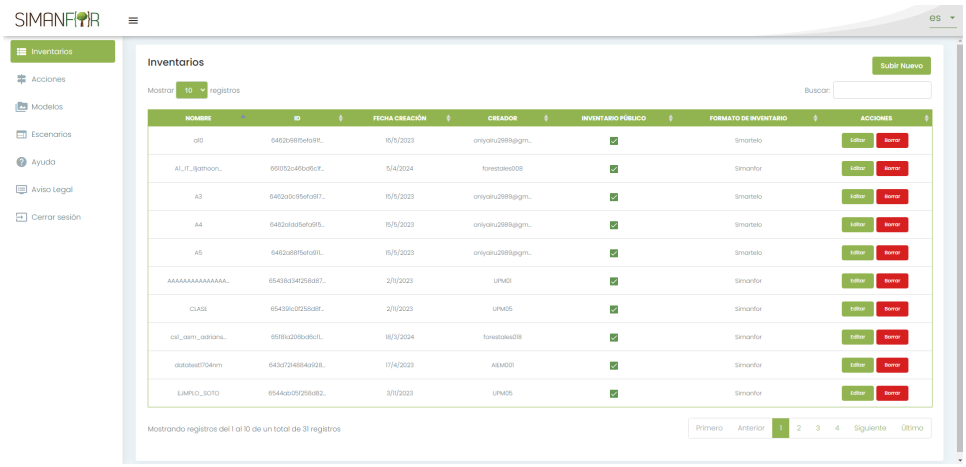

Figura 4.6: Página *inventory* de la web SIMANFOR

- Componente *models* (figura 4.7): Del mismo modo que los anteriores, presenta una tabla en la que se listan los modelos existentes en la base de datos. Se incluye el botón para añadir un nuevo modelo, además de los correspondientes para editar o eliminar uno existente. Estas acciones solo pueden ser realizadas por usuarios administradores, aunque esta validación se realiza en el *backend*.

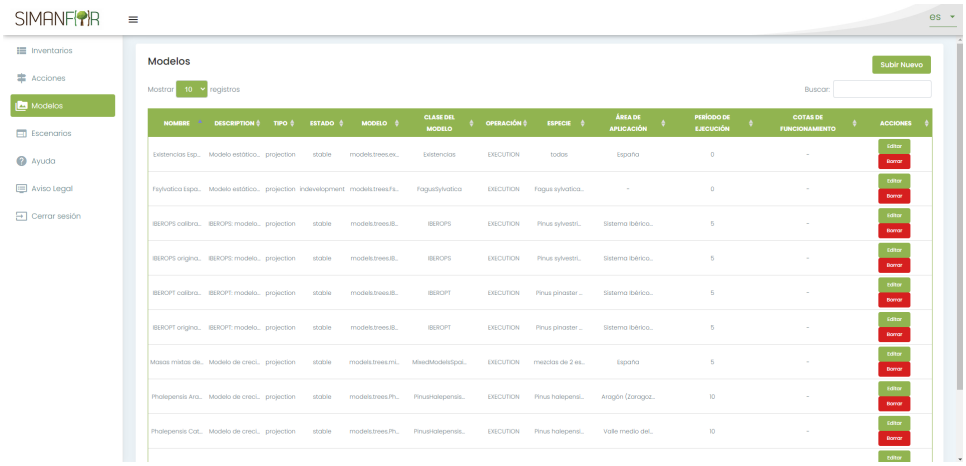

Figura 4.7: Página *models* de la web SIMANFOR

- Componente *scenarios* (figura 4.8): En él también se muestra una tabla, donde se presentan todos los escenarios de la base de datos a los administradores, y solamente los propios a los usuarios básicos. Presenta alguna diferencia con respecto a los anteriores: en primer lugar, el botón para añadir un nuevo escenario está en el HTML del propio componente, en lugar de tomar el del componente *table*, y tras pulsarlo se abre el componente *addScenario*. Por otro lado, al botón de eliminar un escenario se le añaden dos nuevos botones en las filas de la tabla. El primero de ellos abre el componente *addScenario* para poder lanzar la simulación del escenario, y el segundo descarga los resultados de dicha simulación (solamente se muestra una vez terminada la simulación).

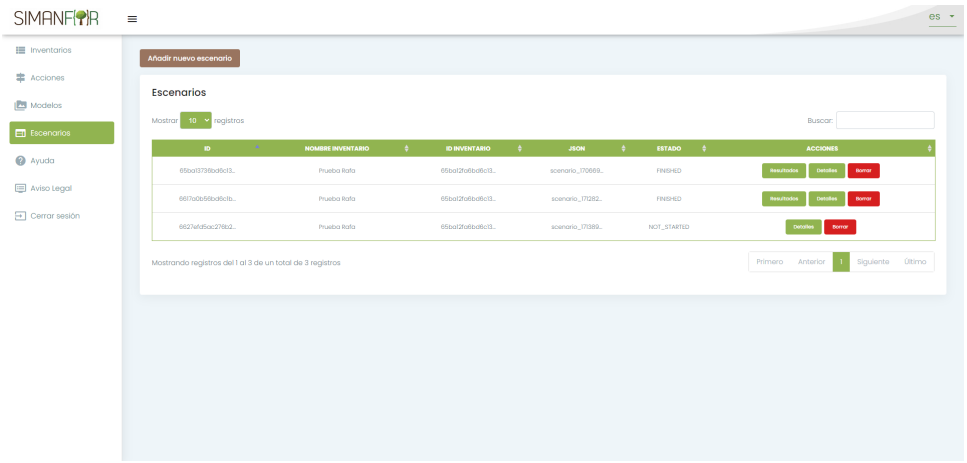

Figura 4.8: Página *scenarios* de la web SIMANFOR

Servicios del frontend

Los servicios de una aplicación Angular se utilizan para dar apoyo al manejo de la lógica del negocio, realizando funciones como la comunicación con el *backend*. En esta aplicación se definen los siguientes servicios, algunos asociados directamente con ciertos componentes de la web:

- Servicio *apiService*: Facilita la comunicación con la API Rest del *backend* de la aplicación, utilizando solicitudes HTTP para ello. En él se definen métodos que configuran los cuatro tipos de solicitudes HTTP: GET, POST, PUT y DELETE. Además, se proporciona soporte para el manejo de las respuestas asíncronas recibidas.

- Servicio *commonService*: Ofrece funcionalidades comunes en toda la aplicación, como el estado del indicador de carga, la obtención de la configuración del idioma o la conversión de un objeto JavaScript en FormData para enviar datos de formularios en las solicitudes HTTP.

- Servicios *authService* y *authGuardService*: Gestionan la autenticación de usuarios en la web. Para ello se almacenan un token y el rol de cada usuario en el almacenamiento local de la aplicación (localStorage), permitiendo así el acceso al resto de funcionalidades y páginas de la web. Estos elementos se crean cuando el usuario inicia sesión en la web, y se eliminan cuando cierra sesión en el sitio o pasadas 24 horas de la creación del token.

- Servicio *usersService*: Proporciona métodos para la gestión de las consultas HTTP al *backend* relacionadas con los usuarios, indicando las rutas de dichos métodos en la API Rest. Se define una consulta GET para obtener todos los usuarios, un POST para agregar uno nuevo y un DELETE para eliminar un usuario existente.

- Servicio *inventoryService*: Se encarga de definir las consultas HTTP relacionadas con los inventarios. Se definen dos consultas GET (una de ellas específica para los inventarios del simulador Smartelo), dos POST (para agregar y editar un inventario) y un DELETE.

- Servicio *modelService*: Gestiona la comunicación con el *backend* relacionada con los modelos. Las solicitudes HTTP de este archivo son tres GET (para obtener todos los modelos, modelos de proyección o modelos de corta), dos POST (para añadir y editar) y un DELETE.

- Servicio *scenarioService*: Tiene la misma función que los anteriores, pero en este caso en relación con los escenarios de la web. Se definen dos consultas GET (una para obtener todos los escenarios y otra para recibir el archivo de resultados a descargar), dos POST (uno para añadir un escenario y otro para lanzar una simulación) y un DELETE.

Tras analizar en detalle el contenido de los repositorios de la web SIMANFOR, se ha establecido un marco conceptual sólido que abarca la estructura y funcionalidades de los distintos componentes y servicios que componen la web. Con este análisis completo, se establece una base sólida para adentrarse en la explicación del desarrollo del proyecto y sus diferentes etapas. Este desarrollo comprenderá una etapa inicial de puesta en marcha del entorno de trabajo local, seguido de los objetivos definidos con ayuda del cliente para conseguir la funcionalidad de la nueva versión del sitio web. Estos objetivos son los siguientes:

- Conexión directa de los resultados de una simulación con la aplicación RShiny.
- Cambio de ubicación del botón *Cerrar Sesión*.
- Reorganización de los datos mostrados en las tablas de la aplicación.
- Enlace a la documentación de los modelos.
- Automatización del proceso de registro de un nuevo usuario en la web.

Capítulo 5

Herramientas de Desarrollo

En esta sección se presentan las herramientas utilizadas en el desarrollo del proyecto. Estas herramientas incluyen tanto el software utilizado por el equipo de SIMANFOR en la versión original de la web como las herramientas específicas empleadas en este proyecto para la consecución de los objetivos propuestos.

5.1 Microsoft Teams

Microsoft Teams [19] es una aplicación creada por Microsoft para gestionar la comunicación entre los miembros de un equipo de trabajo de forma rápida y sencilla. Permite chatear, compartir archivos e integrar software de otras aplicaciones para facilitar la organización del trabajo. En este proyecto, se utilizó para realizar la comunicación con los tutores, sirviendo como medio para resolver dudas, gestionar el progreso y llevar a cabo reuniones telemáticas.

5.2 Trello

Trello [20] es una herramienta de gestión de proyectos que permite dividir el desarrollo en tareas para facilitar el seguimiento del progreso. Estas tareas se organizan en tableros y se clasifican en pendientes, en curso o completadas. En este proyecto, se utilizó Trello al inicio para organizar las fases y tareas de manera visual, y durante el desarrollo para marcar las tareas completadas en cada incremento de progreso.

5.3 Figma

Figma [21] es una herramienta en línea que permite crear maquetas (mock-ups) para el diseño de interfaces de usuario. Permite crear tableros con bocetos más o menos detallados, utilizados principalmente en el diseño de páginas web. Gracias a su almacenamiento en la nube, se empleó en este proyecto para diseñar los mock-ups de las páginas a modificar, como son los relativos a la conexión con RShiny de la figura 8.9 y la figura 8.10, y el del botón *Cerrar Sesión* de la figura 8.22.

5.4 Github

GitHub [22] es un sistema de repositorios en línea para almacenar el código fuente de proyectos. Facilita el trabajo en paralelo de diferentes miembros de un equipo, permitiendo la creación de copias de un mismo repositorio si es necesario. Este sistema es el elegido para almacenar el código fuente de SIMANFOR, de modo que se pudo realizar una copia de los repositorios necesarios para trabajar con ellos de manera individual, en paralelo al desarrollo de la web original.

5.5 Astah Professional

Astah Professional [23] es una herramienta utilizada para la creación de diagramas UML (Unified Modeling Language). Incluye herramientas para la realización de las fases de análisis y diseño de proyectos software, permitiendo la creación de diversos tipos de diagramas. En este proyecto se utilizó para el desarrollo de estas fases, realizando así el diagrama con los actores del sistema de la figura 6.1, el diagrama de casos de uso de la figura 6.2, el diagrama de despliegue de la figura 7.2, el diagrama de clases de la figura 7.3 y el diagrama entidad relación de la figura 7.4.

5.6 Microsoft Word

Microsoft Word [24] es una aplicación de procesamiento de textos incluida en el paquete de Microsoft Office. Permite la redacción, visualización e impresión de documentos. En este proyecto, se utilizó como bloc de notas para registrar los progresos, permitiendo así mantener un control sobre las actividades realizadas y los pasos seguidos en cada una de ellas.

5.7 Overleaf

Overleaf [25] es una aplicación web que guarda documentos en la nube, permitiendo su edición en cualquier momento. Además, permite la creación de documentos de alta calidad mediante el uso de comandos avanzados. La memoria de este Trabajo de Fin de Grado se realizó utilizando esta herramienta, empleando LaTeX como lenguaje de programación orientado a la generación del texto.

5.8 LaTeX

LaTeX [26] es un sistema de composición de textos orientado a la creación de documentos de alta calidad. Se basa en la utilización de comandos, lo que lo convierte en una herramienta muy práctica y útil para este tipo de trabajo. En este proyecto, se utilizó LaTeX para la escritura de la memoria del Trabajo de Fin de Grado a través del procesador de textos en línea Overleaf.

5.9 MobaXterm

MobaXterm [27] es una herramienta para cómputo remoto que proporciona un conjunto de funciones para trabajar en servidores remotos, utilizando una interfaz simple. En este proyecto, se utilizó para la conexión con la máquina virtual en la que se desarrolla el proyecto, mediante SSH desde una máquina Windows a la máquina virtual con Ubuntu como sistema operativo.

5.10 Visual Studio Code

Visual Studio Code [28] es un editor de código fuente con soporte para depuración, control integrado de Git, resaltado de sintaxis, autocompletado, fragmentos de código y refactorización. Además, permite la instalación de extensiones y la conexión a equipos remotos. En este proyecto, se utilizó como método de conexión con la máquina virtual, permitiendo la creación y edición de los archivos necesarios para el desarrollo del proyecto. También se emplearon extensiones para la gestión del ecosistema Docker, el control de versiones en GitHub y la redirección de puertos para acceder a localhost en la máquina virtual.

5.11 Ubuntu

Ubuntu [29] es una distribución GNU/Linux basada en Debian, que incluye en su mayoría software libre y de código abierto. Está orientada al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Este fue el sistema operativo instalado en la máquina virtual sobre la que se desarrolló el proyecto, específicamente una versión de servidor Ubuntu 22.04, con la instalación adicional de Docker.

5.12 Docker

Docker [30] es un proyecto de código abierto orientado a la automatización del despliegue de aplicaciones mediante el uso de contenedores de software. Estos contenedores proporcionan una capa adicional de abstracción y automatización para la virtualización de aplicaciones. El proyecto SIMANFOR se despliega utilizando Docker, con cuatro contenedores: uno para la base de datos, otro para el *backend*, otro para el frontend, y uno más para un proxy que gestiona las conexiones entrantes a la web. Además, se añadió un quinto contenedor para lanzar una instancia de RStudio Server.

5.13 MongoDB

MongoDB [31] es un sistema de base de datos NoSQL orientado a documentos y de código abierto. En lugar de almacenar los datos en tablas, como en las bases de datos relacionales, MongoDB utiliza estructuras de datos BSON (similar a JSON) con un esquema dinámico, lo que facilita y acelera la integración de los datos en ciertas aplicaciones. SIMANFOR utiliza este sistema como base de datos, con cinco colecciones principales: usuarios, acciones (relacionadas con el simulador Smartelo), inventarios, modelos y escenarios.

5.14 Node.js

Node.js [12] es un entorno de ejecución multiplataforma, de código abierto, basado en el lenguaje de programación JavaScript. Es asíncrono, con E/S no bloqueante y una arquitectura orientada a eventos. Este entorno se utiliza como base para la creación del *backend*, debido a su enfoque en maximizar el rendimiento y la eficiencia de las aplicaciones web.

5.15 Angular

Angular [13] es un framework para aplicaciones web desarrollado en TypeScript, de código abierto y mantenido por Google. Se utiliza para crear aplicaciones SPA (Single Page Application), donde se muestra una sola página y las vistas se intercambian dinámicamente. Dada la naturaleza de la web SIMANFOR, este framework se utilizó para desarrollar el frontend de la aplicación, específicamente con Angular 9.

5.16 Nginx

Nginx [8] es un servidor web que también puede funcionar como proxy inverso, balanceador de carga y proxy para protocolos de correo. Es un software ligero, de alto rendimiento, libre, de código abierto y multiplataforma, cuya función principal es gestionar las peticiones HTTPS que llegan a una web. En SIMANFOR, se utiliza un proxy Nginx que recibe las peticiones entrantes y las redirige según la ruta proporcionada.

5.17 Javascript y Typescript

JavaScript [32] es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Es el lenguaje utilizado en el *backend* de la aplicación, debido a su facilidad para crear aplicaciones web y servicios web.

Por otro lado, TypeScript [33] es un superconjunto de JavaScript que añade tipos estáticos y objetos basados en clases. Su principal ventaja sobre JavaScript es la detección de errores de tipos antes de la ejecución, lo que

hace de TypeScript el lenguaje ideal para el desarrollo del frontend de una aplicación web, como es el caso de SIMANFOR.

5.18 HTML y CSS

HTML [34] es el lenguaje de marcado utilizado en la creación de páginas web. Este estándar sirve de referencia para el software que interactúa con la elaboración de páginas web en sus diferentes versiones. Es el lenguaje utilizado para crear las interfaces de usuario (o vistas en el patrón MVVM) de la aplicación web de este proyecto.

CSS [35] (Cascading Style Sheets) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado (habitualmente HTML). Se utiliza en este proyecto para modificar el formato y la presentación de elementos en los archivos HTML, como botones o cuadros de texto.

5.19 Rocker

Rocker [36] es un proyecto que proporciona contenedores Linux para diferentes usos, a partir de imágenes definidas sobre software popular y librerías preinstaladas y optimizadas. En este proyecto, se utiliza la imagen rocker/verse [37], que contiene RStudio Server ya instalado y permite ejecutar una instancia de este software en el ecosistema Docker de la aplicación.

5.20 R

R [38] es un entorno y lenguaje de programación enfocado en el análisis estadístico. Es uno de los más utilizados en investigación científica, debido a sus capacidades de aprendizaje automático, minería de datos e inferencia estadística, entre otras. Además, permite cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo y graficación. En este proyecto, se utiliza R para realizar cálculos sobre los resultados de una simulación, utilizando posteriormente una aplicación Shiny para mostrarlos gráficamente.

5.21 Shiny

Shiny [39] es un paquete de R de código abierto que proporciona un framework elegante y potente para el desarrollo de aplicaciones web usando R. Permite convertir los cálculos resultantes del análisis hecho con R en una aplicación interactiva sin necesidad de utilizar otro tipo de software como HTML, CSS o JavaScript. En este proyecto, se utiliza Shiny para mostrar los resultados de una simulación de forma gráfica, utilizando tablas y gráficos.

5.22 ChatGPT

ChatGPT [40] es una aplicación de chatbot de inteligencia artificial desarrollada por OpenAI, especializada en el diálogo. Es un modelo de lenguaje ajustado con técnicas de aprendizaje tanto supervisadas como de refuerzo. En este proyecto, se utilizó principalmente para la comprensión del código, especialmente en la fase inicial de ingeniería inversa del código de la web original.

5.23 Microsoft Excel

Microsoft Excel [41] es un programa que permite editar hojas de cálculo. Cuenta con funcionalidades de cálculo, gráficas, tablas dinámicas y un lenguaje de programación macro llamado Visual Basic para Aplicaciones. Este es el tipo de documento que proporciona SIMANFOR cuando el usuario solicita la descarga de los datos de una simulación.

Capítulo 6

Análisis

En el análisis del sistema se incluirá la definición de los actores que lo utilizan, la especificación de los requisitos y un diagrama con los casos de uso a realizar. Dado que este proyecto consiste en una nueva versión de una web ya existente, solo se incluirá en el análisis lo relacionado con la nueva funcionalidad añadida o con las funcionalidades ya existentes que se ven modificadas durante el desarrollo.

6.1 Actores del sistema

Los actores se definen como entidades externas al sistema cuya función principal es interactuar con él, ya sea para conseguir objetivos a partir de los servicios del sistema o para proporcionar servicios externos. Es importante definir estos actores para establecer un punto de partida para el resto de los elementos del análisis. Para este proyecto, se identifican cuatro actores (tres principales y uno de apoyo), como se muestra en la Figura 6.1:

- Usuario: Uno de los actores principales; es una persona que accede e interactúa con el sistema.
- Básico: Otro de los actores principales; se trata de un usuario que utiliza la aplicación para realizar simulaciones sobre entornos forestales.
- Administrador: El tercer actor principal; es una persona que accede a la aplicación para realizar simulaciones sobre entornos forestales y para gestionar el sistema.
- Servidor de Simulación: Este es un actor de apoyo que proporciona el servicio de cálculos necesarios para obtener las simulaciones sobre entornos forestales.

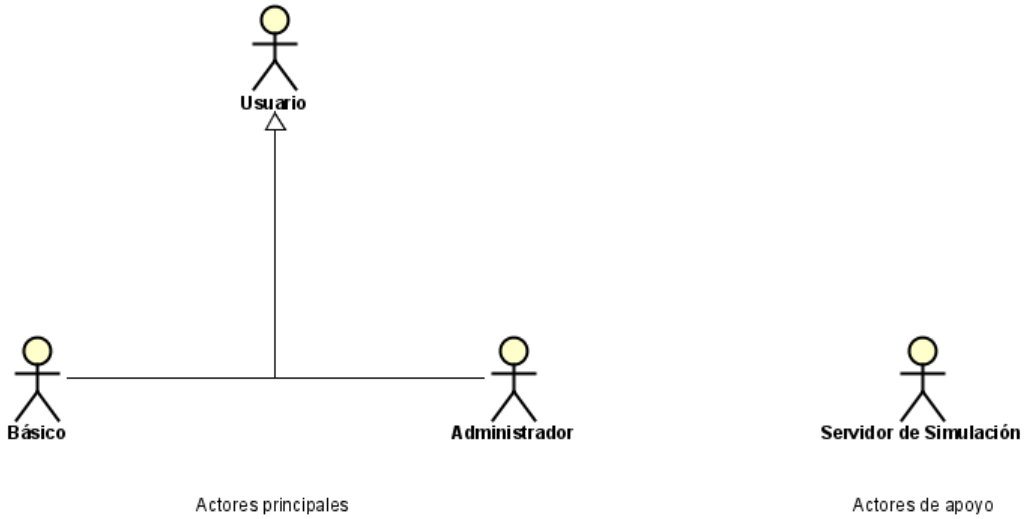

Figura 6.1: Actores del sistema

6.2 Requisitos

En esta sección se especifican los requisitos que el sistema debe cumplir, utilizando como apoyo para su definición tanto los objetivos como el alcance del proyecto. Estos requisitos se dividen en tres categorías: funcionales, no funcionales y de información, cada una detallada a continuación.

6.2.1 Requisitos funcionales

Los requisitos funcionales describen el comportamiento del sistema en términos de los servicios que debe proporcionar, alineados con los objetivos del proyecto. La tabla 6.1 presenta estos requisitos ordenados según fueron definidos dichos objetivos.

Nº	**Descripción**
RF-01	El sistema deberá permitir a los usuarios realizar simulaciones de escenarios utilizando un inventario y un modelo de proyección.
RF-02	El sistema deberá permitir a los usuarios descargar el archivo con los resultados de una simulación.
RF-03	El sistema deberá permitir a los usuarios abrir en el mismo navegador una instancia de RStudio Server.
RF-04	El sistema deberá permitir a los usuarios ver los resultados de una simulación de manera gráfica en la aplicación RShiny.
RF-05	El sistema deberá permitir a los usuarios comparar los resultados de dos o más simulaciones diferentes en la aplicación RShiny.
RF-06	El sistema deberá permitir a los usuarios cerrar sesión en la aplicación.
RF-07	El sistema deberá permitir a los administradores acceder al listado con los usuarios activos.
RF-08	El sistema deberá permitir a los usuarios acceder al listado de escenarios.
RF-09	El sistema deberá permitir a los usuarios acceder al listado de inventarios.
RF-10	El sistema deberá permitir a los usuarios editar los datos de un inventario.
RF-11	El sistema deberá permitir a los usuarios acceder al listado de modelos.
RF-12	El sistema deberá permitir a los administradores editar los datos de un modelo.
RF-13	El sistema deberá permitir a los usuarios abrir en otra pestaña del navegador la documentación de un modelo.
RF-14	El sistema deberá permitir a los usuarios iniciar sesión en el sistema.
RF-15	El sistema deberá permitir a los usuarios generar una solicitud de registro en el sistema.
RF-16	El sistema deberá permitir a los administradores acceder a las solicitudes pendientes de registro de nuevos usuarios.
RF-17	El sistema deberá permitir a los administradores aceptar una solicitud de registro de un nuevo usuario.
RF-18	El sistema deberá permitir a los administradores rechazar una solicitud de registro de un nuevo usuario.

Tabla 6.1: Especificación de requisitos funcionales

6.2.2 Requisitos no funcionales

Los requisitos no funcionales definen propiedades emergentes del sistema, así como las necesidades técnicas o legales que este debe cumplir. Estos requisitos se detallan en la tabla 6.2.

Nº	Descripción
RNF-01	El sistema deberá permitir a un usuario aprender el proceso para visualizar los resultados en la aplicación RShiny en menos de 15 minutos.
RNF-02	El sistema deberá responder de un modo fiable las 24 horas del día, exceptuando durante las actualizaciones del entorno de producción.
RNF-03	El sistema deberá responder en un tiempo inferior a 2 segundos.
RNF-04	El sistema deberá utilizar MongoDB como sistema gestor de base de datos.
RNF-05	El sistema deberá permitir a cualquier ordenador con un navegador web compatible con HTML5 y JavaScript acceder a la aplicación.
RNF-06	El sistema deberá presentar versiones completas de la aplicación en los siguientes idiomas: español, inglés, francés, euskera, gallego, portugués y vietnamita.
RNF-07	El sistema deberá almacenar las contraseñas de los usuarios cifradas en la base de datos.

Tabla 6.2: Especificación de requisitos no funcionales

6.2.3 Requisitos de información

Los requisitos de información especifican el tipo de datos que debe ser almacenado en el sistema; se presentan en la tabla 6.3.

Nº	Descripción
RI-01	El sistema deberá almacenar de un usuario su id, nombre de usuario, contraseña, rol, nombre, apellidos, estado, centro, departamento, email y número de teléfono.
RI-02	El sistema deberá almacenar de un inventario su id, id del inventario principal, nombre, año de registro, id del autor, nombre del autor, fecha de creación, visibilidad (público o privado), formato (Simanfor o Smartelo), nombre del fichero asociado y formato del fichero asociado (xlsx o csv).
RI-03	El sistema deberá almacenar de un modelo su id, nombre, descripción, tipo, enlace a la documentación, estado, ruta al fichero, clase, id del autor, tipo de operación, especie, área de aplicación, período de ejecución y cotas de funcionamiento.
RI-04	El sistema deberá almacenar de un escenario su id, id del inventario, pasos, nombre del fichero JSON, estado, id del autor, nombre del fichero de resultados, id del trabajo en cola, clase del modelo y nombre del modelo.

Tabla 6.3: Especificación de requisitos de información

6.3 Casos de uso

Los casos de uso representan las interacciones entre los actores y el sistema para lograr objetivos específicos. Se utilizan para capturar los requisitos funcionales de manera más detallada y comprensible, mostrando cómo los actores interactúan con las distintas funcionalidades del sistema. A continuación, se presenta en la imagen 6.2 el diagrama de casos de uso del proyecto.

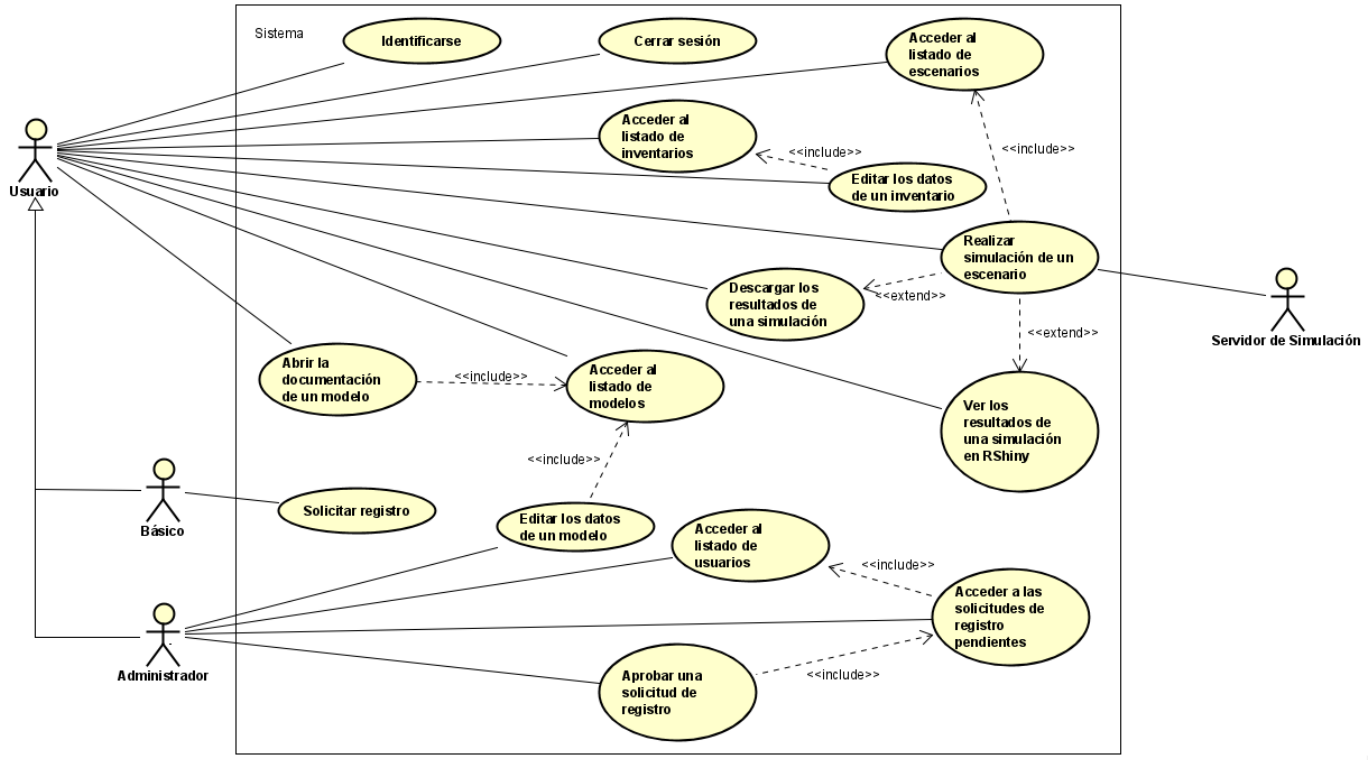

Figura 6.2: Diagrama de Casos de Uso

Capítulo 7

Diseño

7.1 Arquitectura del sistema

Para definir el diseño de la aplicación, se comienza con la especificación de su arquitectura, originalmente creada por el equipo de SIMANFOR al inicio del desarrollo de la web. La aplicación web está desplegada sobre un ecosistema Docker, adoptando así una arquitectura basada en micro-servicios. En un análisis más detallado, se utiliza el patrón MVVM (Model-View-ViewModel) para definir la estructura y la interacción de los componentes dentro de la arquitectura de software. A continuación se presentarán ambas arquitecturas, definiendo los elementos presentes en cada una de ellas antes del inicio del proyecto. Esto sirve como punto de partida para el desarrollo de las nuevas funcionalidades, respetando y siguiendo ambas arquitecturas.

7.1.1 Arquitectura basada en micro-servicios

La arquitectura basada en micro-servicios define un enfoque donde se descompone la aplicación en contenedores independientes, cada uno encargado de una funcionalidad específica. La principal ventaja de este modelo es que se permite una alta escalabilidad, mantenibilidad y flexibilidad [42]. Los componentes de la arquitectura de micro-servicios de esta aplicación son:

- Contenedor de Base de Datos MongoDB: Gestiona el almacenamiento y la recuperación de datos.
- Contenedor de *backend* (Node.js): Proporciona una API Rest que maneja las solicitudes HTTP, interactúa con la base de datos y envía datos al frontend.
- Contenedor de frontend (Angular): Gestiona la presentación y la lógica del lado del cliente, comunicándose con el *backend* a través de servicios donde se definen las consultas HTTP.
- Contenedor de Proxy: Actúa como intermediario, manejando el tráfico entrante y distribuyéndolo a los servicios correspondientes.

- Contenedor de RStudio: Contenedor añadido para la nueva versión de la web, contiene una instancia de RStudio Server para lanzar esta aplicación sin salir del entorno del sistema.

7.1.2 Patrón MVVM

La arquitectura interna de la aplicación web sigue el patrón MVVM (Model, View, View-Model), un patrón de diseño que separa la interfaz de usuario (UI) de la lógica de negocio y de presentación. Esta separación facilita el desarrollo y mantenimiento del código, permitiendo una mayor modularidad y reutilización. Se definen tres componentes: el modelo, la vista y el modelo de vista.

- El modelo representa la lógica de negocio y la gestión de datos, y contiene las entidades que permiten guardar la información en el sistema, recuperar los registros necesarios y gestionar este almacenamiento. En esta aplicación, el modelo está constituido por el *backend* desarrollado en Node.js y la base de datos MongoDB. El *backend* se encarga de realizar las operaciones CRUD (crear, leer, actualizar y eliminar registros) y aplicar las reglas de negocio necesarias antes de enviar los datos al frontend.
- La vista es responsable de la presentación de la interfaz de usuario y de la interacción del usuario. En otras palabras, constituye una representación del modelo en un momento concreto. En la aplicación Angular del frontend, los archivos HTML y CSS de los componentes definen cómo se muestra la información. La vista se centra únicamente en la representación visual, actualizándose dinámicamente según los datos proporcionados por el modelo de vista.
- El modelo de vista maneja la lógica de presentación y actúa como intermediario entre la vista y el modelo. Su función principal es proporcionar los datos y comportamientos necesarios para la vista, realizando la separación entre lógica de presentación y de negocio. En el Angular del frontend, esto se logra mediante dos elementos: los servicios y los archivos Typescript de la lógica de componentes. Los servicios realizan las llamadas a la API Rest del *backend*, gestionando la comunicación con el modelo y suministrando los datos necesarios a los componentes (que describen las páginas web). Los archivos Typescript de los componentes contiene la lógica de presentación y manipulan los datos antes de pasarlos a la vista, recibiendo datos a través de los servicios y realizando transformaciones si es necesario antes de actualizar la vista.

Este enfoque MVVM proporciona una clara separación de responsabilidades, lo que facilita el mantenimiento y la evolución de la aplicación. Al mantener la lógica de negocio, la lógica de presentación y la interfaz de usuario separadas, se mejora la capacidad de prueba y modularidad del sistema [43]. Este patrón de arquitectura se puede ver detallado en la imagen 7.1.

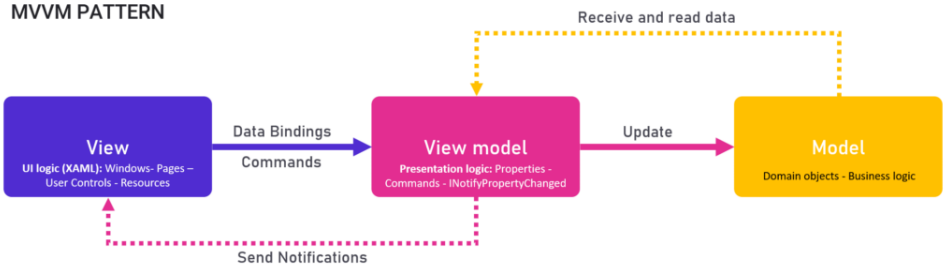

Figura 7.1: Patrón MVVM de arquitectura de software
Fuente: https://rjcodeadvance.com/inicio-de-sesion-en-wpf-implementando-el-patron-mvvm-c-y-sql-server/

7.2 Diagrama de despliegue

Un diagrama de despliegue es un tipo de diagrama UML que proporciona una representación visual de la arquitectura física de un sistema. En él se muestra cómo se distribuyen los componentes de software en los distintos nodos de hardware y cómo se comunican entre ellos. El objetivo principal de este diagrama es proporcionar una comprensión clara y detallada de cómo se organiza la infraestructura de la aplicación, resaltando las conexiones y dependencias entre los diferentes elementos del sistema. Este conocimiento es esencial para el mantenimiento, escalabilidad y mejora continua de la aplicación.

En la figura 7.2 se presenta el diagrama de despliegue de la nueva versión de la aplicación web, detallando la disposición de los contenedores Docker y las interacciones entre ellos. En primer lugar, se muestra el dispositivo del usuario (su ordenador), el cual realiza peticiones HTTP para conectar con el servicio de la aplicación. Esta conexión se realiza a través del servicio de proxy, encargado de redirigir las peticiones entrantes al frontend de la aplicación, desarrollado en Angular. El frontend, a su vez, se comunica mediante solicitudes HTTP con la API Rest del *backend*, que finalmente se conecta con la base de datos MongoDB para el acceso y tratamiento de datos. Además, se ha incluido un nuevo servicio, RStudio Server, que permite al usuario utilizar una instancia de este servicio para ejecutar la aplicación Shiny y acceder a los resultados gráficos de las simulaciones. La seguridad de este tipo de arquitectura se basa en gran medida en el servicio proxy, el cual impide que el usuario acceda directamente a otros servicios de la aplicación, garantizando así un entorno más seguro y controlado.

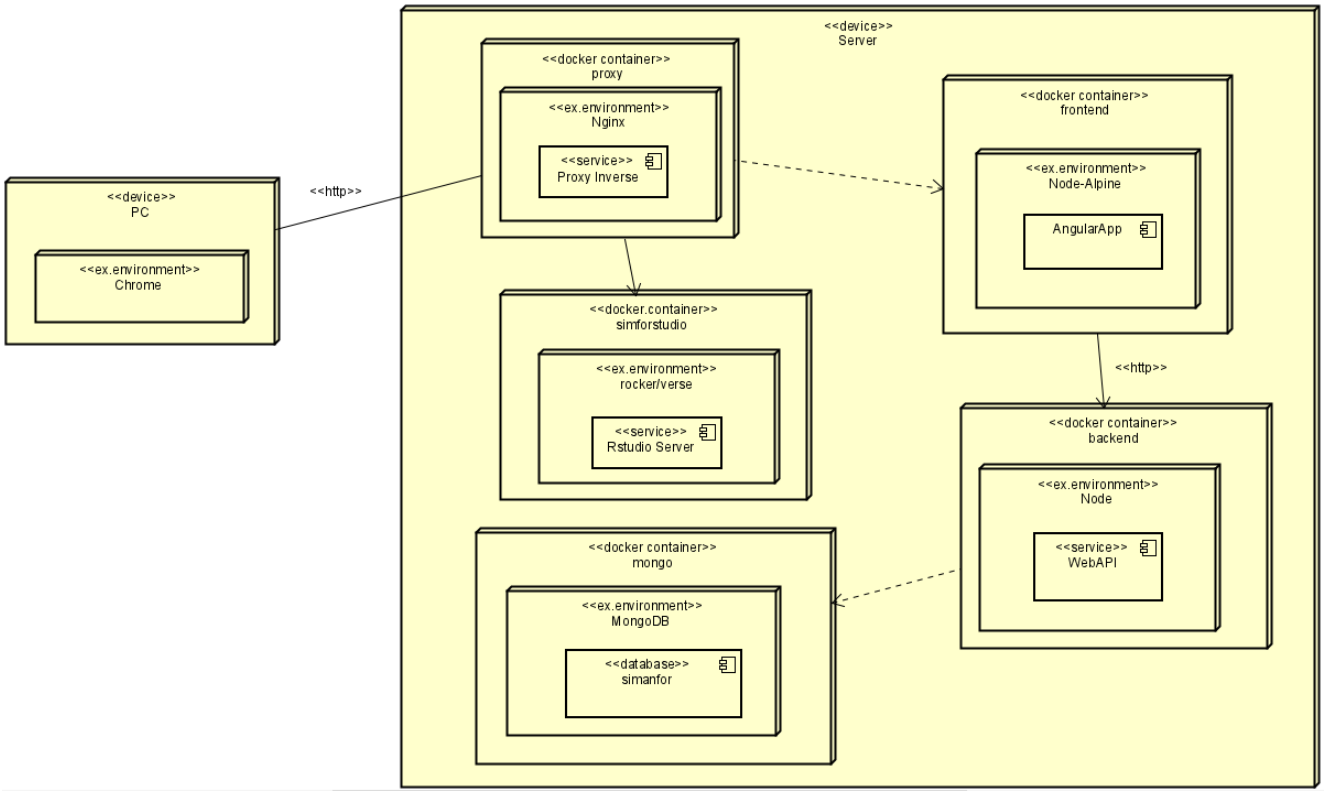

Figura 7.2: Diagrama de despliegue de la nueva versión del sistema

7.3 Modelo de dominio

El modelo de dominio es una representación conceptual del sistema que captura las entidades clave y sus relaciones dentro del dominio del problema. Este modelo proporciona una comprensión detallada de la estructura y el comportamiento del sistema desde una perspectiva de negocio, facilitando la comunicación entre desarrolladores y clientes. A través de diagramas de clases y descripciones detalladas, el modelo de dominio ayuda a garantizar que todos los aspectos críticos del sistema sean correctamente identificados y entendidos, sirviendo como base para el diseño e implementación de la solución. Como en apartados anteriores, el modelo de dominio de la figura 7.3 representa la nueva versión de la aplicación, sumando a la versión anterior las nuevas entidades y relaciones añadidas. Además, se debe aclarar que las entidades representadas solo hacen referencia al simulador SIMANFOR, ya que en el alcance del proyecto se definió que no se tendría en cuenta la funcionalidad de Smartelo.

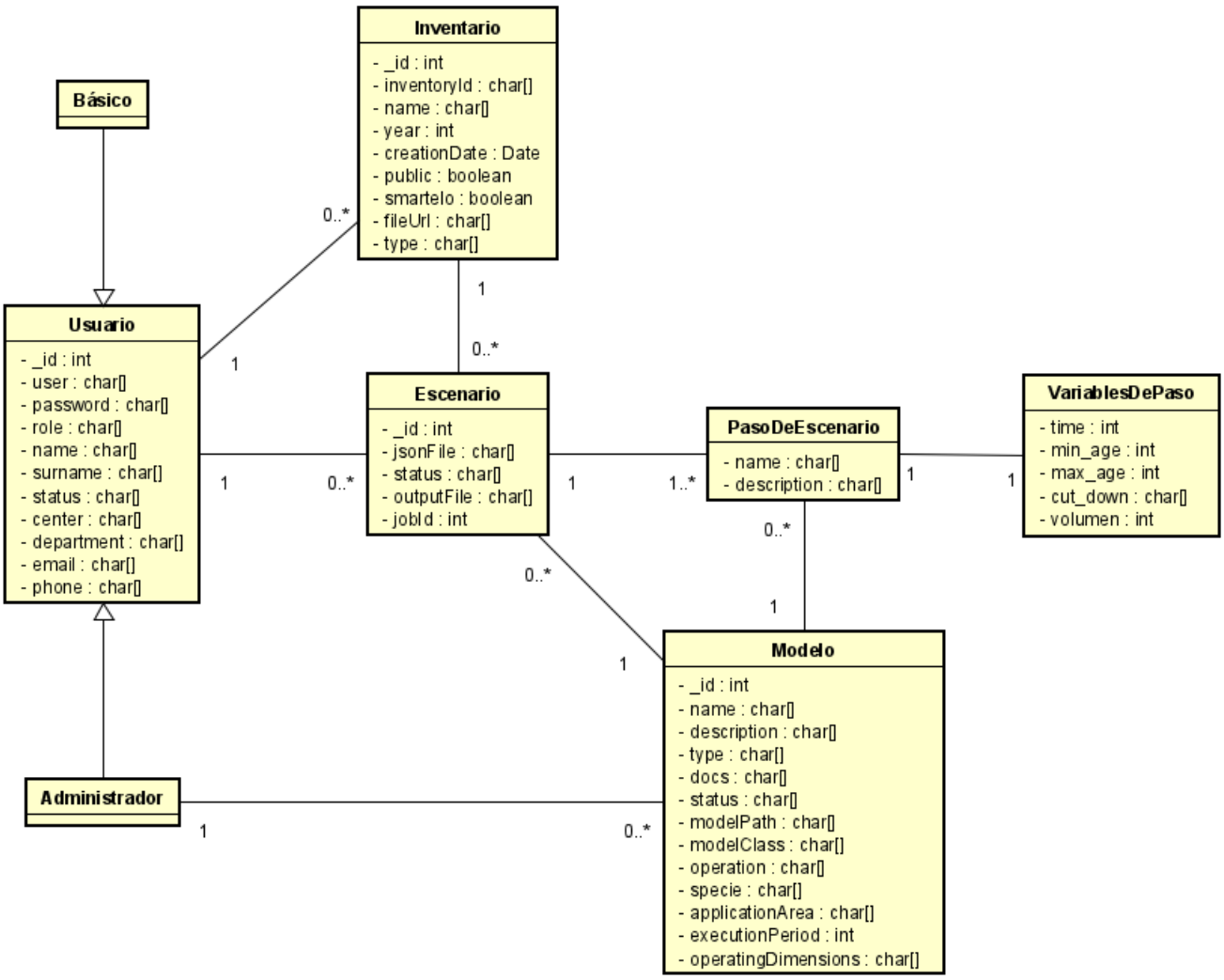

Figura 7.3: Diagrama de clases que representa el modelo de dominio

7.4 Modelo entidad relación

El modelo entidad relación es un diagrama utilizado para representar la estructura lógica de una base de datos de manera gráfica. Este modelo destaca las entidades relevantes del sistema, los atributos de estas entidades y las relaciones entre ellas. En la figura 7.4 se presenta el diagrama relativo a este proyecto, donde se proporciona una visión detallada de cómo se organizan los datos en la base de datos. La base de datos utilizada para este proyecto consiste en una copia de la utilizada en el entorno real de producción, principalmente para que esta no se viera afectada por las modificaciones y pruebas durante el desarrollo del proyecto.

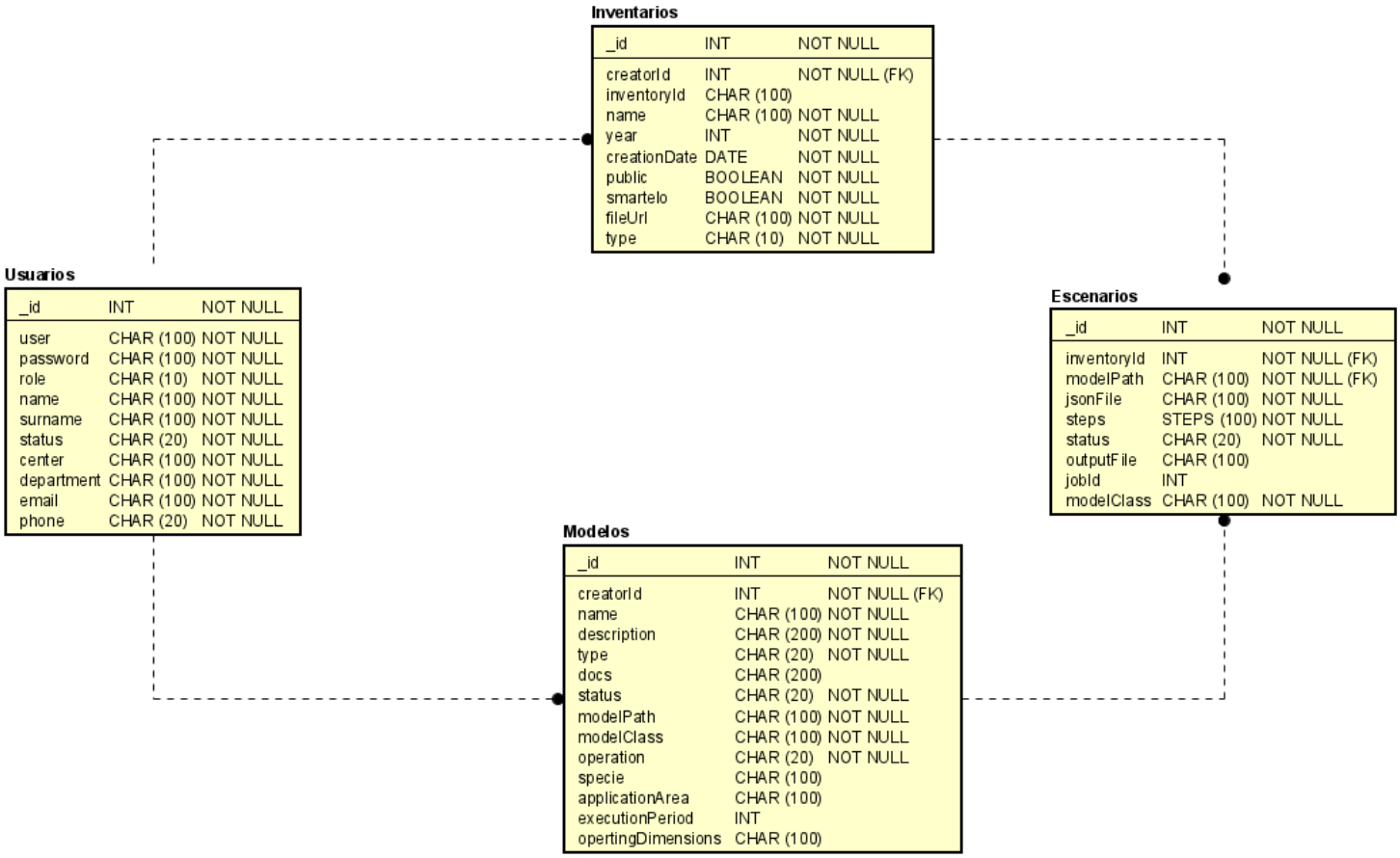

Figura 7.4: Diagrama entidad relación de la base de datos

Capítulo 8

Implementación

Este proyecto consiste en la creación de una nueva versión de la web SIMANFOR (SIMANFOR v2), centrada en mejorar su interfaz de usuario y optimizar la experiencia del usuario. Al término del proyecto, se entregará la versión final al equipo de SIMANFOR para que realicen la migración de las nuevas funcionalidades añadidas.

A continuación, se va a presentar en esta sección el proceso de desarrollo de los distintos objetivos planteados al inicio del proyecto. Cada objetivo se aborda de manera independiente, asegurando que al finalizar cada uno de ellos se cuenta con un servicio completamente funcional. Además, se detalla la preparación del entorno de trabajo como primer paso fundamental para el desarrollo del proyecto.

8.1 Preparación del entorno de trabajo

Para facilitar el desarrollo y pruebas del proyecto, se trabaja en una máquina virtual que permite disponer de todo el entorno de manera local. Esta máquina virtual utiliza Ubuntu Server 22.04, con Docker instalado, junto con soporte para Docker Compose y Docker Buildx. El acceso a la máquina virtual se realiza mediante dos aplicaciones: MobaXterm, que proporciona una consola de comandos y permite la transferencia de archivos entre la máquina utilizada para el desarrollo y la máquina virtual, y Visual Studio Code (en adelante VSC), que se utiliza como entorno de desarrollo integrado (IDE) a través de su función de conexión remota.

El primer paso de la configuración del entorno es clonar los repositorios de GitHub proporcionados por SIMANFOR. Se realiza un fork de cada repositorio para clonar su contenido, permitiendo así que las modificaciones realizadas no afecten a los repositorios originales. Los repositorios clonados son cuatro: production, web-*backend*, web-frontend y simulator. Estos repositorios se copian a VSC para editar los archivos necesarios. Este proceso se llevó a cabo el 12 de diciembre de 2023, por lo que los cambios realizados en los repositorios originales después de esta fecha no se reflejan en la nueva versión.

Para gestionar el desarrollo y las distintas fases del proyecto, se utilizan tanto tags como branches en GitHub. Las tags sirven para señalar puntos específicos en el historial de desarrollo, proporcionando versiones funciona-

les del código. Las branches permiten desarrollar nuevas funcionalidades de manera independiente, sin afectar la estructura original del proyecto [44]. Se crea una tag inicial (versión 1.0 del proyecto SIMANFOR v2) y una nueva branch (project-preparation) para realizar los cambios necesarios para poner en marcha el proyecto.

El siguiente paso es configurar la estructura Docker, ajustando el archivo docker-compose-local.yml para poner en marcha la estructura del proyecto. Se definen cuatro contenedores para desplegar los micro-servicios (las credenciales se ocultan por cuestiones de privacidad): mongo (Figura 8.1), web-*backend* (Figura 8.2), web-frontend (Figura 8.3) y proxy-inverse (Figura 8.4). Estos contenedores utilizan imágenes locales creadas a partir de los Dockerfiles y definen volúmenes con rutas locales para la persistencia de los datos de la base de datos y el *backend*. Cada vez que se realizan cambios en el código, se utiliza el comando *docker compose up –build* para generar nuevas imágenes y levantar los contenedores actualizados.

```
mongo:
  container_name: mongo
  image: mongo:4.2.8-bionic
  restart: always
  ports:
    # - 8081:8081
    - 27017:27017
  environment:
    MONGO_INITDB_ROOT_USERNAME: [redacted]
    MONGO_INITDB_ROOT_PASSWORD: [redacted]
    MONGO_INITDB_DATABASE: simanfor
  volumes:
    - /home/rafagc/simanfor-rgc/persist/mongodb:/data/db
  networks:
    - simanfor
```

Figura 8.1: Contenedor mongo para trabajo en local

```
web-backend:
  container_name: backend
  image: simanfor-dask/web-backend:latest
  build:
    context: ../web-backend
  restart: always
  ports:
    - 3000:3000
  #Check which port uses Node docker container
  environment:
    SSH_HOST: [redacted]
    SSH_PORT: 22
    SSH_USERNAME: [redacted]
    SSH_PASSWORD: [redacted]
    SCRATCH_PATH: /scratch/simanfor
    INPUT_SCRATCH_PATH: /input
    OUTPUT_SCRATCH_PATH: /output
    MONGO_HOST: [redacted]
    MONGO_DB: simanfor
    JWT_MASTER_TOKEN: [redacted]
    # PRODUCTION: 1
  volumes:
    - /home/rafagc/simanfor-rgc/persist/data_input:/usr/src/app/input
    - /home/rafagc/simanfor-rgc/persist/data_output:/usr/src/app/output
  depends_on:
    - mongo
  networks:
    - simanfor
```

Figura 8.2: Contenedor web-*backend* para trabajo en local

```yaml
web-frontend:
  container_name: frontend
  image: simanfor-dask/web-frontend:latest
  build:
    context: ../web-frontend
  restart: always
  depends_on:
    - web-backend
  networks:
    - simanfor
```

Figura 8.3: Contenedor web-frontend para trabajo en local

```yaml
proxy-inverse:
  container_name: proxy
  image: simanfor-dask/production:latest
  build:
    context: .
  restart: always
  ports:
    - 80:80
    - 443:443
  networks:
    - simanfor
  depends_on:
    - web-frontend
```

Figura 8.4: Contenedor proxy-inverse para trabajo en local

El siguiente paso es configurar la base de datos en local. Utilizando el host indicado en el contenedor mongo del docker-compose (sngular_aia_1_3), este contenedor funciona como una instancia local de la base de datos. Se realiza un mongo dump para clonar una copia de una base de datos proporcionada por SIMANFOR, permitiendo realizar cambios sin afectar a la base de datos real. Esta clonación se lleva a cabo el 16 de diciembre de 2023, por lo que los cambios posteriores en la base de datos real no se ven reflejados en el proyecto. Además, se crean dos nuevos usuarios, uno básico y otro administrador, para acceder a la web y trabajar con ellos durante el desarrollo del proyecto.

El último paso para conseguir el acceso a la web y comenzar a trabajar es configurar la redirección de puertos. Dado que el acceso a localhost se realiza en la máquina virtual y no en el equipo del desarrollador, se utilizan túneles de VSC para redirigir los puertos. Tal y como se indica en el contenedor del proxy, el acceso a la aplicación se realiza a través del puerto 443. Por tanto, utilizando el sistema de túneles de VSC se realiza la redirección accediendo a localhost en un puerto cualquiera del equipo utilizado para el desarrollo local (en este caso el 38504) al puerto 443 del contenedor del proxy. Además, se mapean puertos para el *backend* (3000) y para la base de datos (27017), como se muestra en la Figura 8.5.

Port	Forwarded Address	Running Process	Origin
443	localhost:38504		User Forwarded
3000	localhost:3000		User Forwarded
27017	localhost:27017		Auto Forwarded
38504	localhost:38505		Auto Forwarded
Add Port			

Figura 8.5: Redirección de puertos en VSC para el acceso a la web en local

Al obtener acceso a la web a través de localhost, se identifica un problema: la aplicación se está conectando al entorno de producción en lugar del entorno local, impidiendo el trabajo en local. La solución se encuentra en el directorio *environments* del repositorio *web-frontend*; en él se encuentran dos archivos, cada uno de los cuales define la URL para realizar la conexión con el *backend* en un entorno distinto. En la figura 8.6 se puede

ver a la izquierda el archivo *environment.ts*, que define el entorno local, y a la derecha el archivo *environment.prod.ts* que define el entorno de producción. Para cambiar entre estos entornos, se modifica la línea 8 del archivo *package.json* del mismo repositorio. En la figura 8.7 se muestra el contenido de la línea 8: *"build": "ng build"*. En caso de querer cambiar al entorno de producción, se debe escribir en esta línea *"build": "ng build –configuration=production"*.

Figura 8.6: Contenido de los archivos *environment.ts* y *environment.prod.ts*

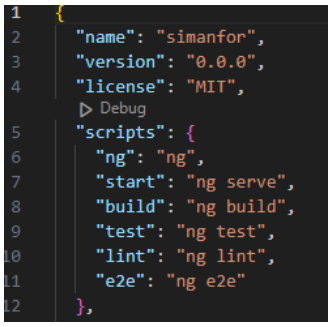

Figura 8.7: Archivo *package.json* del directorio *web-frontend*

Tras realizar estas modificaciones, el entorno queda configurado para trabajar en local sin afectar al entorno de producción. Esta fase del desarrollo experimenta retrasos significativos debido a los problemas encontrados durante la configuración del entorno, lo que provoca que el desarrollo de la implementación comience más tarde de lo esperado.

8.2 Implementación de los objetivos

8.2.1 Conexión de los resultados con la aplicación RShiny

Este objetivo se define durante las reuniones iniciales del proyecto y se aborda como el primer y principal objetivo a implementar. Para explicar de qué se trata, se va a utilizar la figura 8.8; en ella se observa el listado de los escenarios creados por un usuario básico. Se pueden ver dos escenarios ya simulados, indicados con el estado *FINISHED*. Al pulsar el botón *Resultados* de su misma columna, se descarga un archivo .zip con los resultados de la simulación, el cual contiene una hoja de cálculo Excel con los resultados desglosados.

Por otro lado, en el repositorio *Simulator* se encuentra un archivo llamado *Shiny_completo.R*, que permite convertir los datos del archivo Excel en una aplicación Shiny interactiva utilizando el lenguaje de programación R para el cálculo de variables. Sin embargo, este proceso requiere que el usuario realice una gran cantidad de trabajo: debe descargar el archivo .zip, tener instalado un software compatible con R (como RStudio), y ejecutar el archivo *Shiny_completo.R* tras modificar la ruta del directorio de trabajo.

Todos estos pasos hacen que la experiencia de usuario al utilizar la aplicación disminuya considerablemente, debido a la gran cantidad de trabajo que el usuario debe realizar para conseguir su objetivo. Para mejorar esto, se

propone conectar directamente la vista de la Figura 8.8, que contiene el listado de escenarios, con la aplicación RShiny. Además, la aplicación se abrirá en otra pestaña del mismo navegador, permitiendo además al usuario permanecer en todo momento en el ecosistema de la web.

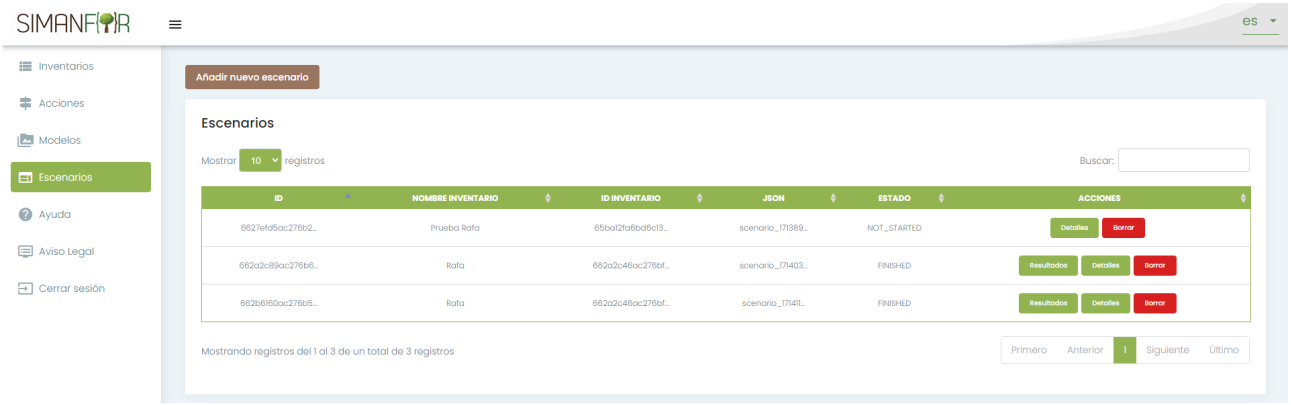

Figura 8.8: Listado de escenarios de un usuario previo al inicio del proyecto

El primer paso para realizar este objetivo consiste en la creación de mock-ups para mostrar las posibles soluciones. Se proponen dos opciones:

- En la figura 8.9 se añade un nuevo botón a la columna *ACCIONES*, presentando dos modelos de texto para dicho botón. La ventaja de esta solución reside en la facilidad para su implementación, ya que solo hay que añadir un botón a los ya existentes. Sin embargo, desde el punto de vista del usuario se observa una sobrecarga de botones en pantalla, lo que puede derivar en una peor experiencia en el uso de la web.

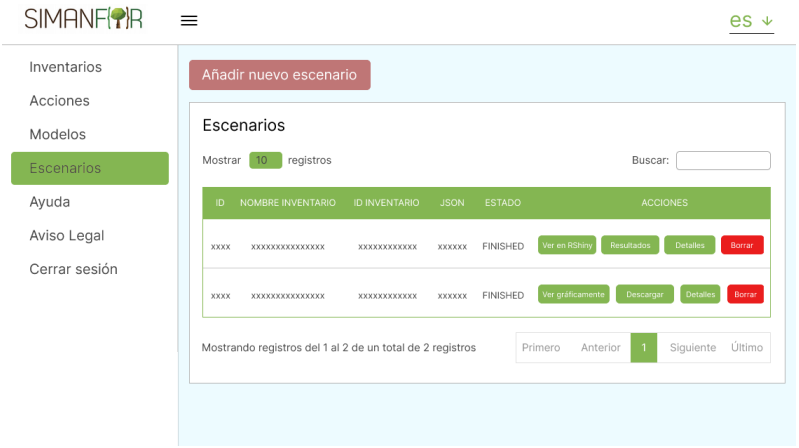

Figura 8.9: Mock-up con la primera opción de solución de conexión con RShiny

- En la figura 8.10 se mantiene la misma estructura del listado de escenarios, pero se añade un nuevo componente que se despliega al pulsar el botón *Resultados*. El punto a favor de esta solución implica mejorar la experiencia de usuario, ya que este componente facilita la comprensión de la información que se observa en pantalla. Por este motivo, se decide llevar a cabo esta solución, debido a que este proyecto busca maximizar la experiencia de usuario en la web.

Figura 8.10: Mock-up con la segunda opción de solución de conexión con RShiny

Para comenzar con la implementación, se añade un nuevo microservicio a la estructura Docker, desplegando un contenedor en el que se ejecuta una instancia de RStudio Server accesible a través de un navegador web. RStudio es un IDE para R que permite ejecutar archivos sin abandonar el entorno Docker ni instalar software adicional en la máquina del usuario.

La figura 8.11 detalla el nuevo servicio añadido al archivo *docker-compose.yml*. Para su despliegue se utiliza la imagen rocker/verse [37], una imagen predeterminada obtenida del proyecto Rocker [36]. Además, se especifica un archivo *Dockerfile* para la instalación de ciertos paquetes necesarios. En él se copia el contenido del archivo *install.packages.R* (figura 8.12) al interior del contenedor de RStudio, y se ejecuta para instalar estos paquetes solamente en caso de que no se hayan instalado previamente en el contenedor (necesario para impedir sobrecarga en el almacenamiento). En la definición del servicio se detalla el puerto 8787 donde escuchará el contenedor, y se definen variables de entorno para permitir la ejecución en modo administrador sin necesidad de autenticación. Por último, se definen dos volúmenes: el primero define la persistencia de los archivos de ejecución de la aplicación R, mientras que el segundo almacena los archivos de resultados de las simulaciones de la web SIMANFOR.

```
rstudio:
  container_name: simforstudio
  image: rocker/verse
  build:
    context: ./rstudio
  restart: always
  ports:
    - 8787:8787
  environment:
    - PASSWORD=[redacted]
    - ROOT=true
    - DISABLE_AUTH=true
  volumes:
    - /home/rafagc/simanfor-rgc/persist/rstudio:/home/rstudio
    - /home/rafagc/simanfor-rgc/persist/data_output:/home/rstudio/data
  networks:
    - simanfor
```

Figura 8.11: Contenedor *simforstudio* con el microservicio de RStudio Server

```
# Lista de paquetes necesarios
packages <- c("plyr", "openxlsx", "plotly")

# Función para verificar e instalar paquetes
install_if_missing <- function(pkg) {
    if (!require(pkg, character.only = TRUE)) {
        install.packages(pkg, repos = "https://cran.rstudio.com/")
    }
}

# Verificar e instalar cada paquete
sapply(packages, install_if_missing)
```

Figura 8.12: Archivo *install.packages.R* para instalar paquetes de RStudio

Tras añadir el microservicio, se debe añadir el nuevo componente mostrado en el mock-up de la figura 8.10. Debido a que no se trata de una página web, si no de un componente que se muestra como desplegable dentro de una página, se crea dentro del directorio *pages/utils* del repositorio *web-frontend*. Además, se debe incluir su definición en el archivo *pages.module.ts*, donde se definen todos los componentes de la aplicación.

Su vista se puede ver en la figura 8.13. Se trata de un desplegable que se muestra al pulsar el botón *Resultados* del listado de escenarios. En él se muestra el nombre del archivo .zip de los resultados, y un mensaje con instrucciones que se detallarán más adelante. Además, se muestran dos botones: el primero mantiene la funcionalidad del botón *Resultados* original del listado de escenarios, es decir, permite descargar el archivo .zip con la hoja de cálculo Excel contenida en él; el segundo es el encargado de conectar con el contenedor de RStudio Server, para poder ver los resultados de una simulación de manera gráfica.

Figura 8.13: Vista del componente *results*

En cuanto a la lógica de la aplicación, se modifica primero el archivo *scenarios.component.ts* para eliminar la funcionalidad anterior del evento *results()*, que era descargar los resultados. En su lugar, se realiza la apertura del nuevo componente, como se ve en la figura 8.14: se abre el componente pasando una instancia del escenario de la lista cuyo botón *Resultados* se ha pulsado, y cargando de nuevo la tabla con el listado de escenarios tras cerrarlo.

```
results(dataModel): void {
  const dialogRef = this.dialog.open(ResultsComponent, {
    data: {},
  });

  dialogRef.componentInstance.dataModel = dataModel;

  dialogRef.afterClosed().subscribe(() => {
    this.loadTableBody();
  });
}
```

Figura 8.14: Evento *results* del archivo *scenarios.component.ts*

En la figura 8.15 se puede ver cómo se implementan las funciones asociadas a los botones de la figura 8.13. En primer lugar, la función *download()* utiliza el servicio *scenarioService.ts* para recibir del *backend* el archivo .zip descargado, para después llamar a la función *downloadFile()* para realizar la descarga desde el navegador web al equipo del usuario. Por otro lado, la función *connectRShiny()* simplemente abre una conexión en otra pestaña del mismo navegador con el contenedor de RStudio Server.

Estos pasos completan las modificaciones realizadas sobre el frontend de la web. Sin embargo, la ruta especificada en el archivo 8.15 para conectar con RStudio Server no funcionaría en un entorno de producción. El acceso a *http://localhost:8787/* se realiza en un entorno local, pero para acceder al entorno de producción se debe descomentar la línea que dirige a *https://www.simanfor.es/rstudio*. Para que esto funcione, se edita el archivo *nginx.conf* del repositorio *production* añadiendo el fragmento que se muestra en la figura 8.16. En este fragmento, las solicitudes dirigidas a */rstudio/* se redirigen al contenedor *simforstudio* definido en el archivo *docker-compose.yml*, que escucha en el puerto 8787.

```
download(_id): void {
  this.loading = true;

  this.scenarioService.downloadResult(_id).subscribe(
    (data) => {
      this.loading = false;
      this.downLoadFile(data, "application/zip");
      console.log(data);
    },
    (error) => {
      this.loading = false;
      console.log(error);
    }
  );
}

downLoadFile(data: any, type: string) {
  let blob = new Blob([data], { type: type });
  let url = window.URL.createObjectURL(blob);
  let pwa = window.open(url);
  if (
    !pwa ||
    pwa.closed ||
    typeof pwa.closed == "undefined" ||
    this.adBlockerEnabled
  ) {
    alert(
      "Pop-up blocker and Ad blocker can block some features of this webpage. Please disable them and try again."
    );
  }
}

connectRShiny(): void {
  window.open('http://localhost:8787/', '_blank');
  // window.open('https://www.simanfor.es/rstudio', '_blank');
}
```

Figura 8.15: Funciones asociadas a los botones del archivo *results.component.ts*

```
location /rstudio/
{
  proxy_pass http://simforstudio:8787;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection 'upgrade';
  proxy_set_header Host $host;
  proxy_cache_bypass $http_upgrade;
}
```

Figura 8.16: Conexión del entorno de producción con RStudio Server definido en *nginx.conf*

El siguiente paso es encontrar un mecanismo para que, al iniciarse en la nueva pestaña la instancia de RStudio Server, se puedan cargar los cálculos del archivo *Shiny_completo.R* y se ejecute la función *shinyApp()* para iniciar la visualización de los gráficos. Este proceso supone la mayor parte del tiempo dedicado al proyecto debido a su complejidad. A continuación, se explican las ideas propuestas para lograrlo, aunque algunas no hayan podido implementarse.

Primero, se crea un archivo *.Rprofile* [45] para que se ejecute cada vez que se inicia una nueva sesión en RStudio Server. Este archivo de control, ubicado en el directorio raíz, configura opciones y variables de entorno al iniciar el servicio. Su contenido se muestra en la Figura 8.17; la función *setHook()* es crucial, ya que indica que el resto del código se ejecutará al iniciar una nueva sesión de RStudio. Luego, se muestra un mensaje de bienvenida y se configura el entorno para que la aplicación Shiny se abra en una nueva pestaña del mismo navegador usando la función *options(shiny.launch.browser = .rs.invokeShinyWindowExternal)*. A continuación, se ejecuta el archivo *Shiny_completo.R* para almacenar sus cálculos como variables del entorno. Finalmente, se imprimen instrucciones para ejecutar la aplicación Shiny, que se detallarán más adelante.

```
setHook("rstudio.sessionInit", function(newSession) {

  if (newSession) {
    message("Welcome to RStudio ", rstudioapi::getVersion());

    # Show shiny app in a new tab
    if (
      # Make sure that {rstudioapi} is available
      requireNamespace("rstudioapi", quietly = TRUE) &&
      # Returns TRUE if RStudio is running
      rstudioapi::hasFun("viewer")
    ) {
      options(shiny.launch.browser = .rs.invokeShinyWindowExternal)
    }

    print("Loading the file 'Shiny_completo.R'... / Cargando el fichero 'Shiny_completo.R'...")
    source("~/simulator/RShiny/Shiny_completo.R")
    print("'Shiny_completo.R' loaded successfully. / 'Shiny_completo.R' cargado exitosamente.")

    print("[English] To view the results using the graphical Shiny app, type 'shinyApp(ui, server)' into this console.
    Then, select the file whose name matches the zip file to visualize its results. To change the app language to Spanish,
    access the file '~/simulator/RShiny/Shiny_completo.R' and modify the value of the variable 'idioma' to 2.")

    print("[Español] Para ver los resultados con la aplicación Shiny gráfica, escribir 'shinyApp(ui, server)' en esta consola.
    Después, seleccionar el archivo cuyo nombre coincide con el fichero zip para visualizar sus resultados. Para cambiar el idioma de la aplicación a inglés,
    acceder al archivo '~/simulator/RShiny/Shiny_completo.R' y modificar el valor de la variable 'idioma' a 1.")

  }
}, action = "append")
```

Figura 8.17: Contenido del archivo *.Rprofile*

Una vez ejecutado el archivo *Shiny_completo.R* mediante *.Rprofile*, el siguiente paso es iniciar la aplicación Shiny. Para ello, se debe escribir en la consola de RStudio Server la función *shinyApp(ui, server)*. Para reducir los pasos que debe realizar el usuario, se intenta automatizar esta función incluyéndola en el archivo *.Rprofile*; sin embargo, se observa que este archivo no puede ejecutarla. Se prueba entonces crear un script bash con esta función y llamarlo al finalizar el contenido del archivo *.Rprofile*, pero este intento también falla, por lo que se concluye que el usuario debe escribirla manualmente en la consola. Por este motivo, se añaden instrucciones en el archivo *.Rprofile*, disponibles en español e inglés.

El siguiente problema que se afronta es la necesidad de reiniciar la sesión para ejecutar nuevamente el archivo *Shiny_completo.R*. Si el usuario no ha realizado una nueva simulación, no hay problema, ya que los archivos de resultados almacenados son los mismos. Sin embargo, tras simular un nuevo escenario, se debe reiniciar la sesión de RStudio Server para que el archivo *.Rprofile* cargue los nuevos resultados. Tras realizar pruebas modificando este archivo, se concluye que lo mejor es ejecutar el archivo *Shiny_completo.R* al iniciar una nueva sesión, tal como se muestra en la figura 8.17. Por esta razón, se añaden las instrucciones mostradas en la Figura 8.13 al componente *results*.

Por último, se aborda el problema de los archivos de resultados. Cuando se pulsa el botón *Resultados* de uno de los escenarios en la tabla, se abre el componente mostrado en la figura 8.13. Este componente almacena el escenario correspondiente, permitiendo la opción de descargar el fichero de resultados asociado. En el caso de la aplicación RShiny, se intenta pasar también el escenario a RStudio Server para cargar únicamente su archivo en la aplicación. Sin embargo, esto no es posible debido a la falta de comunicación entre contenedores, ya que son contenedores muy básicos con funcionalidad limitada. Por ello, la solución adoptada consiste en utilizar el segundo volumen del contenedor definido en la 8.11. Este volumen asegura que todos los archivos .zip guardados en *data_output* se almacenan en el directorio *data* del contenedor de RStudio Server. De esta manera, todos los archivos de resultados se cargan en la aplicación RShiny a través del archivo *Shiny_completo.R*.

Para lograr esto, se edita el archivo *Shiny_completo.R*, que originalmente solo permitía cargar más de un archivo de resultados si estos estaban en inglés. Así, se toman todos los archivos de resultados de la web y se realizan los cálculos necesarios para que puedan ser mostrados en la aplicación Shiny. La mayoría de las modificaciones se centran en los cálculos en R, añadiendo soporte para archivos en español además de los ya

existentes en inglés.

La figura 8.18 muestra un ejemplo de cómo se visualizan los datos en la aplicación RShiny. En primer lugar, se permite elegir el archivo de resultados que se desea visualizar, tal como se indica en las instrucciones del archivo *.Rprofile* (figura 8.17). Además, el usuario puede elegir entre diferentes contenidos a visualizar, seleccionando entre diversas variables y tipos de gráficos.

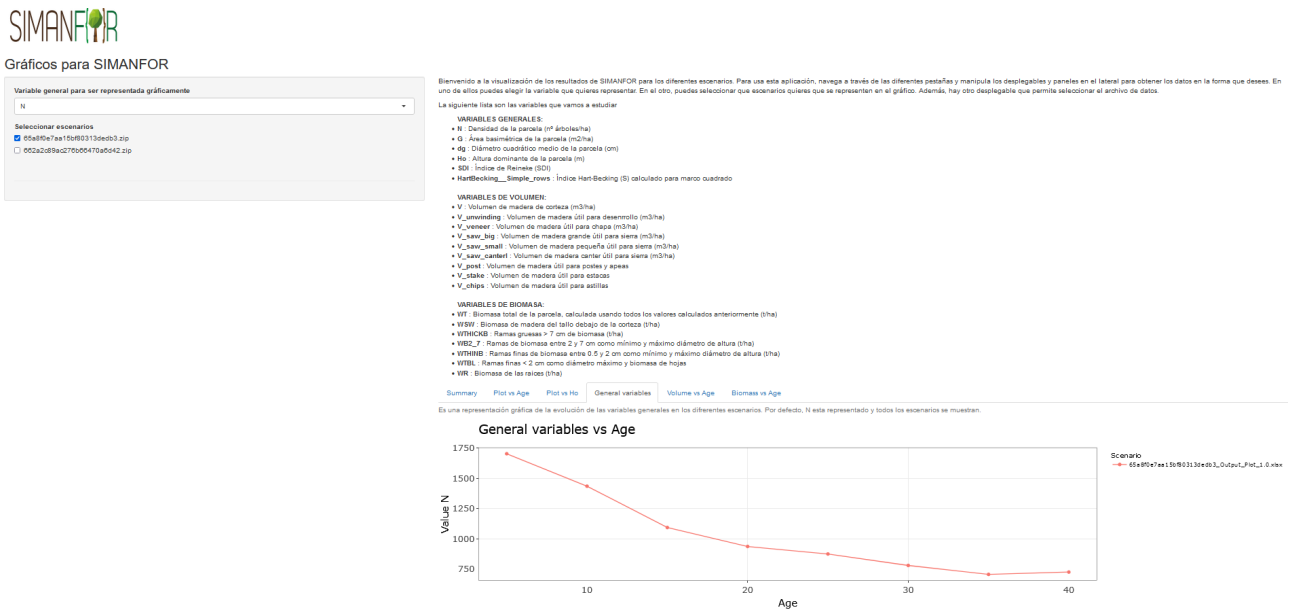

Figura 8.18: Ejemplo de visualización de resultados en la aplicación RShiny

Para realizar las modificaciones mencionadas, hay dos cambios en el código del archivo *Shiny_completo.R* que deben destacarse:

- La figura 8.19 muestra el código modificado para permitir al usuario elegir qué archivo de resultados desea ver en la aplicación Shiny. Se distinguen dos tipos de gráficos: en el primero, el usuario puede elegir uno o más archivos para realizar comparaciones gráficas; en el segundo, solo se puede visualizar un archivo de resultados, seleccionando uno por defecto.

```
conditionalPanel(
  condition = "input.tabset == 'Volume vs Age' || input.tabset == 'General variables' || input.tabset =='Biomass vs Age'",
  checkboxGroupInput("scenarios", lLanguage[[8]][idioma],
                     choices = nombreZips),
),
conditionalPanel(
  condition = "input.tabset == 'Plot vs Ho' || input.tabset == 'Plot vs Age'|| input.tabset =='Summary'",
  selectInput("nombreArchivos",lLanguage[[9]][idioma],
              choices = nombreArchivos,
              selected = nombreArchivos[1])
),
```

Figura 8.19: Fragmento de *Shiny_completo.R* para permitir elegir los resultados a visualizar

- En la figura 8.20 se indica cómo se muestra la leyenda de los gráficos en la aplicación Shiny (esto aplica solo a los gráficos del primer tipo mencionado anteriormente). Primero, se identifican los escenarios representados en el gráfico. Luego, se crea el gráfico asignando un color a cada escenario (usando la función *color = as.character(idNombre)*), y colocando dicha asignación en la leyenda del gráfico (mediante la asignación *color = "Scenario"*).

```
output$graficoGeneral <- renderPlotly({

  # selected_scenarios <- input$scenarios
  # Obtener las posiciones de los valores seleccionados en 'escenarios'
  selected_scenarios <- which(escenarios %in% input$scenarios)
  dfFiltered <- dfMedias[dfMedias$IdEscenario %in% selected_scenarios, ]

  gg<-ggplot(dfFiltered, aes(x = T, y = eval(parse(text = input$variableG)), color = as.character(IdNombre), group = IdNombre)) +
    geom_point() +
    geom_line() +
    labs(title = bquote("General variables vs Age"),
         subtitle = input$variableG,
         x = "Age",
         y = paste0("Value ", input$variableG),
         color = "Scenario") +
    theme_bw()+
    theme(plot.title = element_text(size = 20))+
    theme(axis.text = element_text(size = 14),  # Ajusta el tamaño del texto de los ejes
          axis.title = element_text(size = 16))  # Ajusta el tamaño del título de los ejes
  ggplotly(gg,tooltip = c("x","y"))
})
```

Figura 8.20: Fragmento de *Shiny_completo.R* para mostrar la leyenda de las gráficas correctamente

Con esto finaliza la explicación de la implementación realizada para lograr esta funcionalidad. El objetivo era reducir el esfuerzo y los recursos necesarios para que el usuario visualice los resultados de una simulación en la aplicación gráfica RShiny. Esto se ha logrado principalmente al permitir al usuario trabajar en todo momento en el entorno de la aplicación web, mostrando instrucciones claras y concisas en cada paso. Estos son los nuevos pasos que debe seguir el usuario en la nueva versión:

1. En primer lugar, el usuario pulsa el botón *Resultados* del listado de escenarios, abriéndose el componente mostrado en la figura 8.13.
2. Después, pulsa el botón *Ver en RShiny*, que abre una instancia de RStudio Server en otra pestaña del navegador.
3. Siguiendo las instrucciones del componente anterior, reinicia la sesión de RStudio Server y espera a que el archivo *Shiny_completo.R* realice los cálculos necesarios.
4. A continuación, escribe en la consola de RStudio Server la función *shinyApp(ui, server)*, que abre la aplicación Shiny en otra pestaña del navegador.
5. Por último, el usuario selecciona en la aplicación Shiny el archivo de resultados que quiere visualizar, teniendo en cuenta que el nombre se proporciona en la vista del componente de la figura 8.13.

Sin embargo, es necesario aclarar un detalle acerca de la solución implementada. El uso de una instancia de RStudio Server viene dado porque, aunque se quiere facilitar a los usuarios la visualización de sus resultados en Shiny, se reconoce que R es el lenguaje de programación utilizado por la mayoría de ingenieros forestales a día de hoy. A pesar de que Python se está utilizando cada vez con mayor frecuencia, R sigue siendo el más popular entre los modeladores y estadísticos de la gestión forestal.

Una solución más rápida podría haber sido ejecutar el archivo *Shiny_completo.R* a través de línea de comandos (utilizando el comando *Rscript*), lo que permitiría ver los resultados gráficos solamente al pulsar el botón *Ver en RShiny* de la vista de los resultados. Sin embargo, si el usuario es un ingeniero forestal que está familiarizado con R y quiere jugar con los datos resultantes, producir otros gráficos y/o evaluar estadísticas, tendría que tener instalado RStudio en su máquina local. Esto implicaría hacer el mismo proceso que en la versión anterior de la aplicación, descargando el archivo *Shiny_completo.R* y ejecutándolo a través de la aplicación RStudio de escritorio. Siguiendo la solución implementada, se añade algún paso adicional pero se permite al usuario jugar fácil y eficazmente con el código directamente en la instancia de RStudio Server ejecutada en su contenedor.

El desarrollo de esta funcionalidad constituyó la mayor parte del tiempo dedicado al proyecto, ya que se consideró la base y el objetivo más importante. A continuación, se describen el resto de objetivos, con un mayor enfoque en el frontend de la aplicación y en la mejora de la experiencia del usuario.

8.2.2 Reubicación del botón *Cerrar Sesión*

En la figura 8.21 se puede ver la ubicación original del botón que permite cerrar la sesión. Este se mostraba en el menú lateral, como un elemento más del listado de páginas web presentes en el menú. Esta disposición genera confusión en la interfaz de usuario, ya que la función de este botón no es coherente con la del resto de elementos.

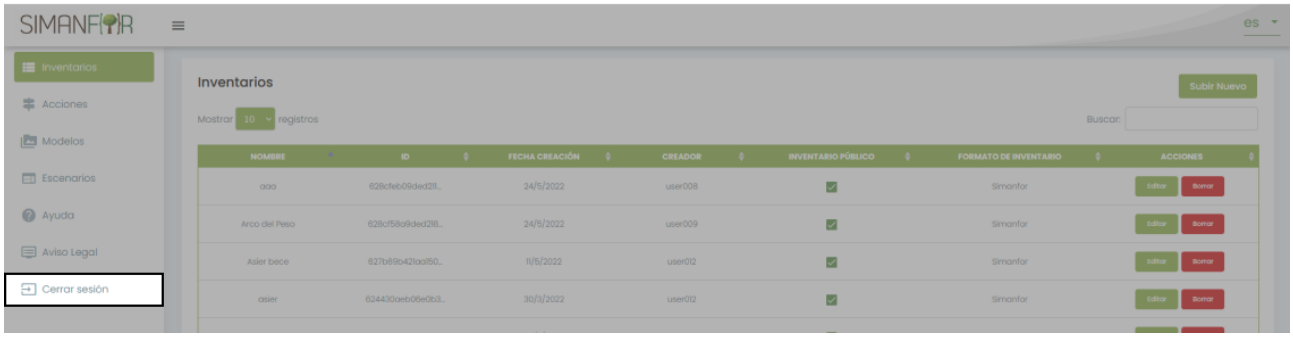

Figura 8.21: Ubicación original del botón *Cerrar Sesión*

Para mejorar la interfaz de usuario, se propone cambiar la ubicación de este botón, tomando como referencia páginas web como YouTube [46] o LinkedIn [47]. En estas páginas, las funciones relacionadas con ajustes y la gestión de la cuenta del usuario autenticado se encuentran en la parte derecha del menú superior. Por tanto, se define esta ubicación como la nueva para el botón *Cerrar Sesión*, además de la opción de selección de idioma que ya estaba presente.

El primer paso consiste en decidir el formato del nuevo botón. Con el objetivo de maximizar la experiencia de usuario, se propone utilizar algún tipo de icono que se relacione visualmente con la funcionalidad buscada. Para ello, se realizan varios mock-ups con diferentes iconos, eligiendo el de la figura 8.22 para su implementación.

Figura 8.22: Mock-up con el que se decide el formato del botón *Cerrar Sesión*

El componente a editar para añadir este botón es *layouts/full/header*. Este componente contiene el código mostrado en el encabezado de la web, que inicialmente solo incluye el botón de selección de idioma. En la figura 8.23 se muestra el nuevo código, manteniendo la selección de idioma y añadiendo el botón de cierre de sesión. Para ello, se utiliza el icono seleccionado en el mock-up anterior; además, se especifica que el botón solo se muestra si el usuario que accede a la web está autenticado, utilizando el servicio *authService* para su comprobación.

```
<div class="header-menu-container">
  <mat-form-field class="lenguage-select">
    <mat-select [value]=translate.currentLang (selectionChange)="switchLang($event)">
      <mat-option *ngFor="let language of translate.getLangs()" [value]=language>{{ language }}</mat-option>
    </mat-select>
  </mat-form-field>

  <button mat-icon-button (click)="closeSession()" class="logout-button" *ngIf="authService.isAuthenticated()" title="{{ 'menu.closesession' | translate }}">
    <img src="assets/images/logout-icon.png" alt="Logout">
  </button>
</div>
```

Figura 8.23: Archivo *header.component.html* donde se muestra el botón *Cerrar Sesión*

En cuanto a la lógica del componente, se edita el archivo *header.component.ts* añadiendo la función mostrada en la figura 8.24. Primero, se abre el componente de confirmación como medida de seguridad (lo que mejora la experiencia de usuario), y tras el cierre de este, se utiliza el servicio *authService* para gestionar el cierre de sesión. Por último, se redirige a la página de inicio de la web.

```
closeSession() {
  const confirmation = this.dialog.open(ConfirmationComponent);
  confirmation.componentInstance.message = "menu.close_session_confirmation";
  confirmation.afterClosed().subscribe((result) => {
    if (result) {
      this.authService.logout();
      this.router.navigate(["/"]);
    }
  });
}
```

Figura 8.24: Archivo *header.component.ts* con la lógica para el cierre de sesión

El último paso consiste en eliminar el código del botón anterior, ubicado en el menú lateral. Este código se encuentra en el componente *layouts/full/sidebar*, del cual se eliminan los fragmentos correspondientes de sus archivos HTML y TypeScript. En la figura 8.25, se observa el resultado de esta implementación: se añade el icono que funciona como botón de cierre de sesión y se elimina el anterior del menú lateral, dejando en él solo enlaces a páginas de la web.

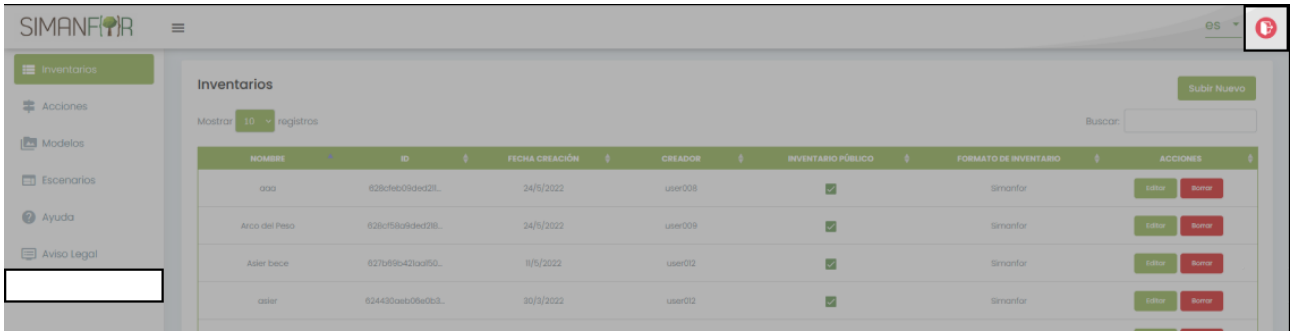

Figura 8.25: Ubicación final del botón *Cerrar Sesión*

8.2.3 Reorganización de los datos mostrados en las tablas

Esta tarea se basa en los listados de elementos que aparecen en la página web. Estos listados son tablas que contienen información sobre los elementos guardados en la base de datos de la aplicación. El objetivo de las modificaciones en este apartado es mostrar al usuario únicamente la información relevante y suficiente que necesita ver cuando accede a estos listados. Además, se analiza la información mostrada al editar uno de estos elementos, modificándola en caso de ser necesario.

La primera tabla a analizar es la de usuarios. A esta tabla solo pueden acceder los usuarios administradores, que son quienes controlan la funcionalidad relacionada con los usuarios en la web. Como en la última columna no se permite editar un usuario, es necesario mostrar todos los datos almacenados en la base de datos en la tabla. Por este motivo, no se realiza ninguna modificación sobre esta tabla, manteniendo el formato que se puede ver en la figura 8.26 (se oculta el número de teléfono por privacidad).

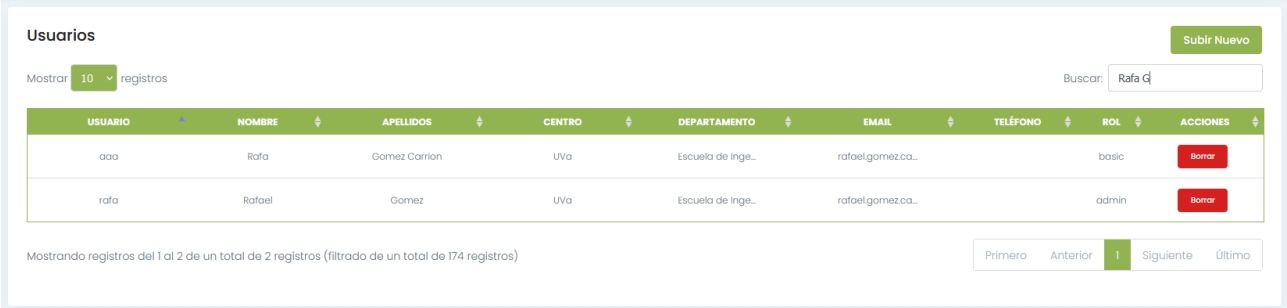

Figura 8.26: Tabla *Usuarios* de su página correspondiente

La siguiente tabla es la de escenarios. En la figura 8.27 se muestra el listado original de este tipo de elementos. En él se presentan datos irrelevantes, como el ID del inventario con el que se ha creado el escenario (con mostrar el nombre es suficiente) y el archivo JSON que se crea para almacenar los pasos en la base de datos. Por este motivo, se decide eliminar estos datos para que no se muestren en el listado.

Figura 8.27: Tabla *Escenarios* previa a las modificaciones

Por otro lado, es necesario mostrar información sobre el modelo de proyección y el autor que ha creado el escenario. Para ello, se modifica el código mostrado en la figura 8.28; en la variable *header*, se añaden el nombre del autor y los campos con el nombre y la clase del modelo de proyección.

```
ngOnInit(): void {
  this.loadTableBody();

  setInterval(() => {
    console.info("Refreshing table data...");
    this.loadTableBody();
  }, 30 * 1000);

  this.header = ["_id", "creatorName", "inventoryName", "modelPath", "modelClass", "status"];
}
```

Figura 8.28: Cabeceras de la tabla de escenarios del archivo *scenarios.component.ts*

El resto de datos de esta tabla no se modifican, mostrando como resultado el aspecto que se ve en la figura 8.29. Además, los datos mostrados al editar un escenario tampoco se modifican, ya que solo se muestra el listado de pasos que lo componen.

Figura 8.29: Tabla *Escenarios* tras las modificaciones

La última tabla de este apartado es la de inventarios. En la figura 8.30 se muestra la tabla en su formato original, recortando algunos elementos para no sobrecargar la imagen. En ella se muestra información relevante, como el nombre, la fecha de creación, el autor, su formato y si es público o no. Sin embargo, el ID del inventario no es necesario mostrarlo, ya que con el nombre es suficiente. Se elimina dicho ID de la tabla, eliminándolo del atributo *header* del archivo *inventory.component.ts*, y obteniendo como resultado la tabla de la figura 8.31.

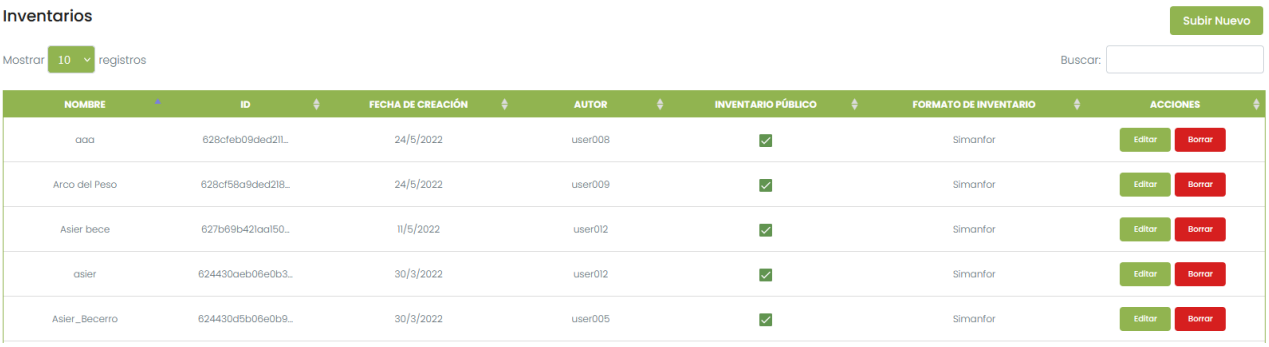

Figura 8.30: Tabla *Inventarios* previa a las modificaciones

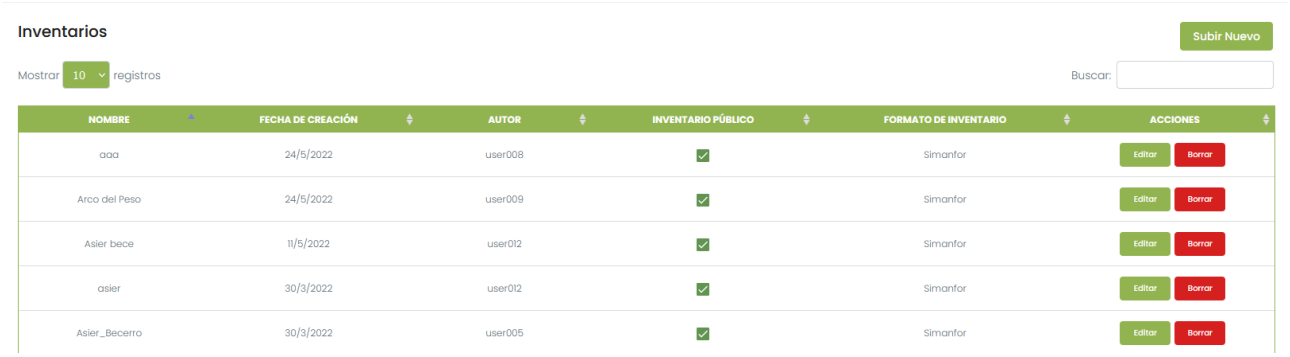

Figura 8.31: Tabla *Inventarios* tras las modificaciones

Por otro lado, cuando se selecciona la opción de editar un inventario, el componente que se abre tiene el formato de la figura 8.32. En él se muestra muy poca información, solo aquella que se puede editar. Además, el título mostrado es erróneo, ya que no se está añadiendo un nuevo inventario, sino editando uno existente.

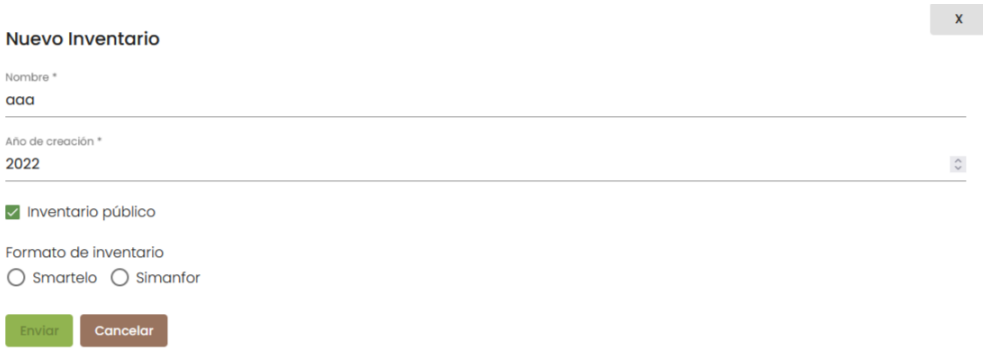

Figura 8.32: Vista del componente de editar un inventario antes de las modificaciones

Para modificar la información de este componente, se modifica el atributo *addForm* del archivo *inventory.component.ts*. Este es el formulario que se envía al componente *addElement* con los datos que deben mostrarse en él, como se ve en la figura 8.32. En este caso, se añaden los tres elementos de la figura 8.33. Para ellos se define un nuevo tipo, *show*, de forma que estos campos del formulario no sean editables y solo sirvan para mostrar información. Este tipo es mostrado por el archivo *add-element.component.html* de la forma que se indica en la figura 8.34. En primer lugar, se comprueba que estos atributos solo se muestren al editar un elemento; después, se añade la cabecera del campo y, por último, se imprime su contenido.

```
{
  name: "_id",
  type: "show",
  required: false,
  editable: false,
},
{
  name: "creator",
  type: "show",
  required: false,
  editable: false,
},
{
  name: "fileUrl",
  type: "show",
  required: false,
  editable: false,
},
```

Figura 8.33: Fragmento del atributo *addForm* del archivo *inventory.component.ts*

```
<div *ngIf="editId !== undefined && field.type === 'show'" style="margin-bottom: 10px;">
  <label style="display: block; font-size: 0.8em;">{{ dataModelName + '.' + field.name | translate }}</label>
  <div style="border-bottom: 1px solid rgb(133, 133, 133); padding-bottom: 5px;">
    <span>{{ data[field.name] }}</span>
  </div>
</div>
```

Figura 8.34: Atributos de tipo *show* del archivo *add-element.component.html*

De esta forma, el resultado final al seleccionar editar un inventario es el que se ve en la figura 8.35.

Figura 8.35: Vista del componente de editar un inventario tras las modificaciones

8.2.4 Enlace a la documentación de los modelos

El objetivo de esta tarea es incorporar en la tabla de modelos un enlace directo a la documentación correspondiente de cada uno. De este modo, el usuario no necesitará acceder a la información del modelo, copiar el enlace y pegarlo en la barra de búsqueda del navegador; simplemente deberá pulsar el botón correspondiente en la fila del modelo cuya documentación desea consultar.

La primera parte de esta tarea implica realizar un análisis similar al de las tablas anteriores. Se examinan los datos mostrados en la tabla original, visible en la figura 8.36. En esta figura se puede observar la presencia de información irrelevante, como la operación o el nombre del archivo del modelo. También se incluye un campo para la descripción, que resulta inapropiado debido a su extensión considerable. Por tanto, se eliminan estos tres campos de la tabla, modificando el atributo *header* en el archivo *models.component.ts*.

Figura 8.36: Tabla *Modelos* previa a las modificaciones

La información eliminada de la tabla sigue siendo accesible cuando se selecciona la opción de editar un modelo. En ese mismo componente, se añade el atributo *ID* de los modelos, configurado como un elemento de tipo *show*. De manera similar a la edición de un inventario, este dato se muestran al usuario sin permitir su edición. El resultado de este componente se ve en la figura 8.37.

Figura 8.37: Vista del componente de editar un modelo tras las modificaciones

La segunda parte del objetivo consiste en incluir el enlace a la documentación de un modelo en su fila correspondiente de la tabla. Para ello, se debe modificar el archivo *table.component.html* como se muestra en la figura 8.38. Este archivo define la estructura general de las tablas que se utilizan en las páginas de usuario, inventarios, modelos y escenarios. Primero, se añade a la cabecera de la tabla el encabezado correspondiente, ya que la documentación no forma parte de los atributos enviados por el atributo *header* de *models.component.ts*, al ser una información especial. Luego, en el cuerpo de la tabla se incluye una palabra clave con un enlace, utilizando el elemento *[href]="row.docs"*. Esto hace que la palabra clave contenga un hipervínculo cuyo valor es la URL almacenada en la base de datos correspondiente a la documentación de ese modelo.

Como resultado, la tabla con el listado de los modelos recibe el formato mostrado en la figura 8.39. En esta figura se puede observar cómo se han eliminado los campos mencionados anteriormente y se ha añadido una nueva columna con una palabra clave que, al ser pulsada, redirige a la documentación del modelo correspondiente.

```
<thead>
<tr>
  <th *ngIf="!!selectable">
    <mat-checkbox (change)="$event ? masterToggle() : null"
                  [checked]="selection.hasValue() && isAllSelected()"
                  [indeterminate]="selection.hasValue() && !isAllSelected()"
                  *ngIf="!!allowMultiSelect">
    </mat-checkbox>
  </th>
  <th *ngFor="let field of tableHeader">{{ tableTitle + '.' + field | translate }}</th>
  <th *ngIf="tableTitle === 'models'">{{ tableTitle + '.docs' | translate }}</th>
  <th *ngIf="rowButtons.length >= 1 && !selectable">{{ 'table.actions' | translate }}</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let row of tableBody">
  <td *ngIf="!!selectable">
    <mat-checkbox (click)="$event.stopPropagation()"
                  (change)="$event ? selection.toggle(row) : null"
                  [checked]="selection.isSelected(row)">
    </mat-checkbox>
  </td>
  <td *ngFor="let field of generateArray(row)">
    <div *ngIf="isBoolean(field)">
      <mat-checkbox [ngModel]="field" disabled></mat-checkbox>
    </div>
    <div *ngIf="!isBoolean(field) && !!field">
      <p title="{{field}}">{{ field | cutter }}</p>
    </div>
  </td>
  <td *ngIf="tableTitle === 'models'">
    <a [href]="row.docs">{{ tableTitle + '.linkDocs' | translate }}</a>
  </td>
```

Figura 8.38: Enlace a la documentación de un modelo en el archivo *table.component.html*

Figura 8.39: Tabla *Modelos* tras las modificaciones

8.2.5 Registro automático de usuario

El objetivo de esta tarea es modificar el proceso de registro de nuevos usuarios. En la figura 8.40 se puede observar la página de inicio de sesión de la web SIMANFOR original. El mensaje en la parte inferior indica que, para crear un usuario, es necesario descargar un formulario, rellenarlo y enviarlo por correo al equipo de SIMANFOR. Posteriormente, el equipo se encargará de validar la petición y crear manualmente el nuevo usuario, utilizando el botón *Nuevo Usuario* del listado de usuarios, visible en la parte superior derecha de la imagen 8.26.

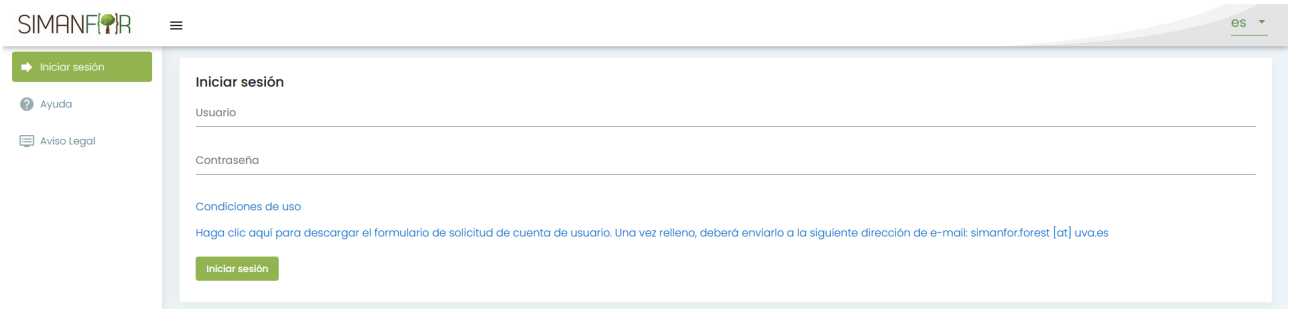

Figura 8.40: Página de inicio de sesión previa a las modificaciones

Este proceso requiere una considerable intervención por parte del usuario, lo que reduce la experiencia de usuario en la web. Para mejorar este aspecto, se busca automatizar el proceso de registro. Se presentará al usuario un formulario para que pueda solicitar la creación de una cuenta, y posteriormente esta solicitud será revisada por los administradores, quienes la validarán o rechazarán, creando la nueva cuenta en caso de validación.

El primer paso consiste en cambiar la ubicación del botón *Iniciar Sesión*. En la figura 8.40 se muestra dicho botón en el menú lateral, junto con el listado de páginas de la web. De forma similar al botón de cierre de sesión, se decide reubicarlo en el menú superior, junto a la selección de idioma. Para ello, se añade al archivo *header.component.html* el código mostrado en la figura 8.41. En este archivo, además del código añadido previamente para mostrar el botón de cierre de sesión, se incorpora el botón *Iniciar Sesión*, que solo se muestra cuando no hay ningún usuario autenticado, y redirige a la página ilustrada en la figura 8.40. Además, se eliminan tanto el mensaje que informaba del proceso anterior como el código del archivo *sidebar.component.html* que mostraba el botón en el menú lateral.

```
<div class="header-menu-container">
  <mat-form-field class="lenguage-select">
    <mat-select [value]=translate.currentLang (selectionChange)="switchLang($event)">
      <mat-option *ngFor="let language of translate.getLangs()" [value]=language>{{ language }}</mat-option>
    </mat-select>
  </mat-form-field>

  <button mat-button class="login-button" color="primary" *ngIf="!authService.isAuthenticated()" title="{{ 'menu.login' | translate }}" [routerLink]="['/login']">
    {{ 'login.title' | translate }}
  </button>
  <button mat-icon-button (click)="closeSession()" class="logout-button" *ngIf="authService.isAuthenticated()" title="{{ 'menu.closesession' | translate }}">
    <img src="assets/images/logout-icon.png" alt="Logout">
  </button>
</div>
```

Figura 8.41: Archivo *header.component.html* donde se muestra el botón *Iniciar Sesión*

El siguiente paso es añadir a la página de inicio de sesión un botón para realizar la solicitud de registro de nuevo usuario. Para ello, se añade al archivo HTML correspondiente el código mostrado en la figura 8.42, donde se incluye un botón que llama a la función *newUserRequest* del archivo TypeScript cuando se pulsa.

```
<hr>

<p style="text-align: center;"> {{ 'login.newUser' | translate }} </p>
<div style="display: flex; justify-content: center;">
  <button mat-button (click)="newUserRequest()" color="primary">{{ 'login.new' | translate }}</button>
</div>
```

Figura 8.42: Botón de solicitud de nuevo usuario del archivo *login.component.html*

La función *newUserRequest* se muestra en la figura 8.43. Su propósito es abrir el componente *AddElementComponent*, indicando que el elemento mostrado es un usuario a través del atributo *dataModelName*. También se pasan los campos del formulario a rellenar, definidos en el atributo *form*. Finalmente, al cerrar el componente, se muestra el componente de confirmación, y si el usuario acepta, se utiliza el servicio *ApiService* para realizar una petición HTTP al *backend* para crear el usuario.

Tras realizar las modificaciones descritas, el aspecto final de la página *Iniciar Sesión* es el que se muestra en la figura 8.44. Se observa la nueva ubicación del botón *Iniciar Sesión* y el nuevo botón para realizar una solicitud de registro, acompañado de un mensaje explicativo.

El siguiente paso es editar el componente *AddElement* que se abre cuando un usuario solicita el registro de una nueva cuenta. Este componente ya estaba previamente definido, pero se añaden dos nuevos campos para el tratamiento de contraseñas. En la figura 8.45 se detallan estos campos. Inicialmente, la contraseña no se muestra, pero se incluye un botón que permite al usuario visualizarla. Además, cuando se modifica el valor de cualquiera de los dos campos, se ejecuta la función de validación.

```
newUserRequest(): void {
  const dialogRef = this.dialog.open(AddElementComponent, {
    data: {},
  });

  dialogRef.componentInstance.dataModelName = 'users';

  let form = [...this.addForm];

  dialogRef.componentInstance.dataModel = form;

  dialogRef.afterClosed().subscribe((result) => {
    // The dialog was closed
    if (result) {
      const confirmation = this.dialog.open(ConfirmationComponent);
      confirmation.componentInstance.message = "table.confirmation_regist_user";
      confirmation.afterClosed().subscribe((res) => {
        if (res) {
          this.apiService.postFormData("/register", this.commonService.objectToFormData(result)).subscribe(
            (resp) => {
              console.log("ADD OK");
              console.log(resp);
            },
            (error) => {
              console.log("ADD KO", error.status);
              this._showSnack(
                "Error: " + this.translate.instant("error." + error.error.string_code)
              );
            }
          );
        }
      })
    }
  });
}
```

Figura 8.43: Función *newUserRequest* del archivo *login.component.ts*

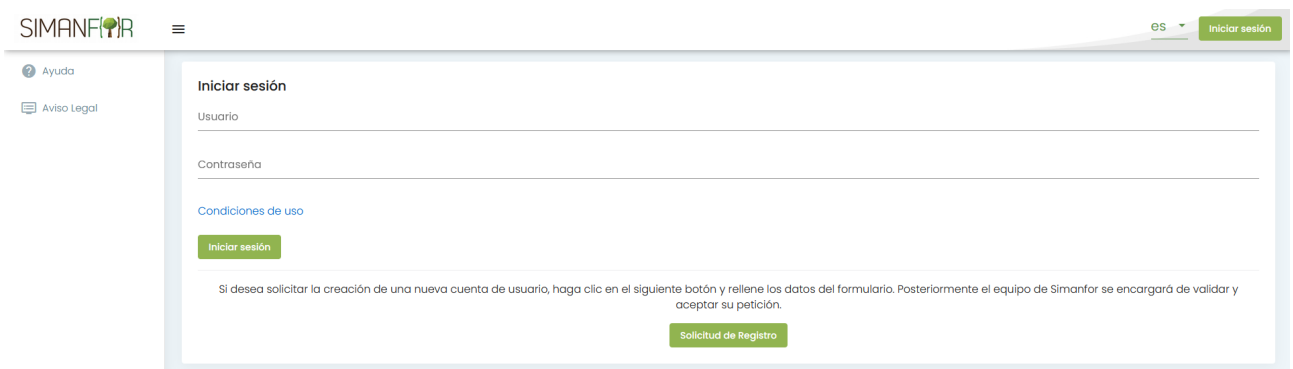

Figura 8.44: Página de inicio de sesión tras las modificaciones

La lógica para tratar las contraseñas se muestra en la figura 8.46. El método *updateValidPassword()* comprueba que los campos de las contraseñas contengan el mismo valor, estableciendo el resultado de la comprobación en la variable *validPassword*. En caso de que no coincidan, se muestra un mensaje de error y no se permite enviar la solicitud de registro. Los otros dos métodos manejan la lógica del botón que oculta el texto de las contraseñas, cambiando su valor cada vez que se pulsa.

```
<mat-form-field *ngIf="field.type == 'password'" class="password-field">
  <mat-label>{{ dataModelName + '.' + field.name | translate }}</mat-label>
  <div class="password-input-container">
    <input name="field.name" matInput tabindex="1" [(ngModel)]="data[field.name]" [type]="passwordFieldType" *ngIf="field.required"
    required (ngModelChange)="updateValidPassword(); updateValidInput()">
    <button mat-icon-button tabindex="-1" (click)="togglePasswordVisibility()"
    [attr.aria-label]="passwordFieldType === 'password' ? 'Show password' : 'Hide password'" class="password-toggle-button">
      <img src="assets/images/view-password.png" alt="Toggle password visibility">
    </button>
  </div>
  <mat-error *ngIf="field.required">
    {{ 'error.required_params' | translate }}
  </mat-error>
</mat-form-field>

<mat-form-field *ngIf="field.type == 'confirmPassword'" class="password-field">
  <mat-label>{{ dataModelName + '.' + field.name | translate }}</mat-label>
  <div class="password-input-container">
    <input name="field.name" matInput tabindex="1" [(ngModel)]="confirmPassword" [type]="confirmPasswordFieldType"
    (ngModelChange)="updateValidPassword(); updateValidInput()">
    <button mat-icon-button tabindex="-1" (click)="toggleConfirmPasswordVisibility()"
    [attr.aria-label]="confirmPasswordFieldType === 'password' ? 'Show password' : 'Hide password'" class="password-toggle-button">
      <img src="assets/images/view-password.png" alt="Toggle password visibility">
    </button>
  </div>
</mat-form-field>
```

Figura 8.45: Código para mostrar contraseñas en el archivo *add-element.component.html*

```
updateValidPassword(): void {
  if (this.data['password'] !== this.confirmPassword) {
    this.validPassword = false;
  } else {
    this.validPassword = true;
  }
}

togglePasswordVisibility() {
  this.passwordFieldType = this.passwordFieldType === 'password' ? 'text' : 'password';
}

toggleConfirmPasswordVisibility() {
  this.confirmPasswordFieldType = this.confirmPasswordFieldType === 'password' ? 'text' : 'password';
}
```

Figura 8.46: Código para gestionar contraseñas en el archivo *add-element.component.ts*

El resultado final del componente de solicitud de registro se muestra en la figura 8.47. Se detallan los campos a rellenar, incluyendo los de confirmación de contraseña. Hasta que no se hayan completado todos los campos obligatorios (marcados con *) y las contraseñas coincidan, no se permitirá enviar la solicitud, mostrando el mensaje correspondiente en caso de error.

Una vez que el usuario ha realizado la solicitud, se cierra el componente y se realiza la consulta HTTP para guardar ese nuevo usuario en la base de datos, como se ilustra en la figura 8.43. El servicio *ApiService* genera dicha petición, una operación *POST* que es recibida por el *backend* en la ruta */register*. En el archivo *auth.js* del *backend* se deben realizar varias modificaciones para implementar este proceso:

- Se añade la ruta */register* al listado de rutas públicas, que son accesibles por un usuario no autenticado. En este caso, al realizar la petición de nuevo usuario, aún no se ha realizado la autenticación.
- Antes de añadir el registro a la base de datos, se verifica que no exista un usuario con el mismo nombre de usuario o correo electrónico, lo que generaría un error. Los usuarios con estado *REFUSED* no se incluyen en esta verificación.
- Se crea una nueva variable aparte de las proporcionadas por el usuario para almacenar el estado en la base de datos. Esta variable se inicializa con el valor *REQUESTED*.

Figura 8.47: Componente de solicitud de nuevo usuario tras las modificaciones

- Se crea el usuario en el sistema, añadiéndolo a la base de datos con los atributos proporcionados por el usuario y el estado *REQUESTED*.
- Para que el listado de usuarios activos solo devuelva aquellos que están aceptados, se añade un filtro en la operación *GET* que devuelve todos los usuarios de la base de datos. En este caso, se verifica que los usuarios devueltos no tengan estado *REQUESTED* ni *REFUSED*.

El último paso consiste en mostrar a los usuarios administradores las solicitudes de nuevo usuario pendientes, para que puedan aceptarlas o rechazarlas. Para ello, se modifica el componente *table*, que muestra el listado de usuarios (figura 8.26); el botón *Subir Nuevo* se modifica por uno que abre el componente con los datos de una solicitud de registro pendiente. Además, en su texto se muestra el número de solicitudes pendientes en ese momento. Para ello, cuando se accede a la página de usuarios, en su método *ngOnInit()* se ejecuta el método mostrado en la figura 8.48. Esta función utiliza el servicio *UsersService* para generar una solicitud HTTP de tipo *GET* que devuelve un listado con los usuarios que tienen el atributo de estado *REQUESTED*. Se almacenan en dos variables tanto dicho listado como su número de elementos, los cuales son pasados al componente *table* para su uso. El segundo se utiliza en el botón mencionado anteriormente, mostrando así el número de solicitudes pendientes y dejando la tabla de usuarios como se ve en la figura 8.49 (los datos de los usuarios se han ocultado por privacidad).

```
getRequestedUsers(): void{
  this.usersService.getRequestedUsers().subscribe(
    (resp) => {
      this.requestedUsers = resp.data;
      this.numRequestedUsers = this.requestedUsers.length;
    },
    (error) => {
      console.error(error.status);
      this._showSnack(
        "Error: " + this.translate.instant("error." + error.error.string_code)
      );
    }
  );
}
```

Figura 8.48: Función *getRequestedUsers()* del archivo *users.component.ts*

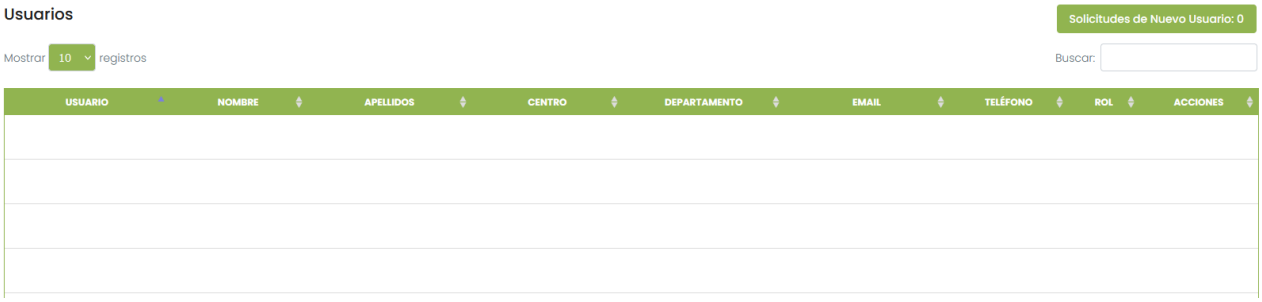

Figura 8.49: Tabla *Usuarios* con el botón de solicitudes pendientes añadido

Cuando un administrador pulsa el botón para ver las solicitudes pendientes, se ejecuta el método ilustrado en la figura 8.50. En primer lugar, se comprueba que haya al menos un usuario pendiente; en ese caso, se abre el componente *AddElementComponent*, al que se le pasa el nombre de la tabla, la posición del usuario que se va a mostrar en la lista de pendientes (inicialmente el primero), su ID, el número total de usuarios pendientes y el formulario con los campos a mostrar (todos los elementos son de tipo *show*).

```
openRequestedUsersDialog():void {
  if (this.numRequestedUsers > 0) {
    const dialogRef = this.dialog.open(AddElementComponent, {
      data: {},
    });

    dialogRef.componentInstance.dataModelName = this.tableTitle;

    let form = [...this.addForm];

    const actualUser = this.requestedUsers[this.numActualUser];

    dialogRef.componentInstance.editId = actualUser._id;
    dialogRef.componentInstance.numRequestedUsers = this.numRequestedUsers;
    dialogRef.componentInstance.numActualUser = this.numActualUser;

    form = form.map((formItem) => {
      if (actualUser[formItem.name]) {
        return (formItem = {
          ...formItem,
          default: actualUser[formItem.name],
        });
      }

      return formItem;
    });

    dialogRef.componentInstance.dataModel = form;
```

Figura 8.50: Parte 1 de la función *openRequestedUsersDialog()* del archivo *table.component.ts*

El contenido del componente que contiene las solicitudes de registro de usuario se muestra en la figura 8.51.

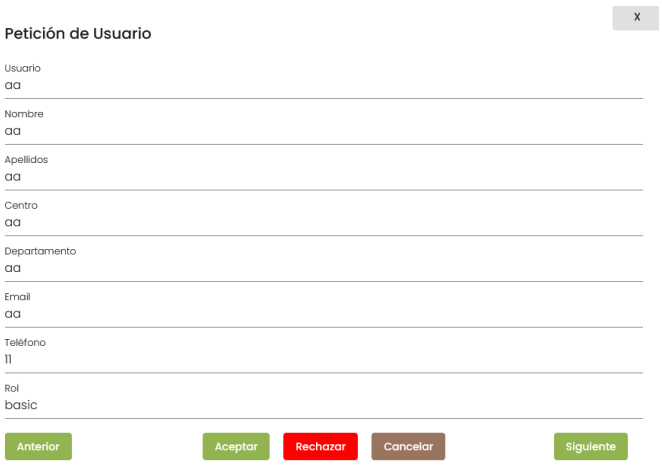

Figura 8.51: Componente *AddElement* con una solicitud de usuario pendiente

En primer lugar, se muestran los campos del formulario recibido, todos en formato *show* para que no puedan ser editados. Además, se presentan varios botones con distintas funcionalidades:

- El botón *Cancelar* cierra el componente y no realiza ninguna acción adicional.
- El botón *Aceptar* hace que se ejecute la segunda parte de la función *openRequestedUsersDialog()*, mostrada en la figura 8.52. Cuando se cierra el componente, primero se comprueba si se ha hecho alguna acción sobre la misma solicitud con la que se abrió el componente (es decir, si se ha aprobado o rechazado). Luego, se abre el componente de confirmación y se emite el evento *addEvent*.

```
dialogRef.afterClosed().subscribe((result) => {
  // The dialog was closed
  if (result) {
    if (result.actualUser == this.numActualUser){
      const confirmation = this.dialog.open(ConfirmationComponent);
      if(result.status === 'REFUSED') {
        confirmation.componentInstance.message = "table.confirmation_refuse_user";
      } else {
        confirmation.componentInstance.message = "table.confirmation_accept_user";
      }
      confirmation.afterClosed().subscribe((res) => {
        if (res) {
          this.numRequestedUsers--;
          console.log("editEvent launched");
          this.addEvent.emit(result);
        }
      });
    } else {
      this.numActualUser = result.actualUser;
      this.openRequestedUsersDialog();
    }
  }
});
```

Figura 8.52: Parte 2 de la función *openRequestedUsersDialog()* del archivo *table.component.ts*

- El botón *Rechazar* realiza las mismas acciones que el botón anterior, pero antes de cerrar el componente de solicitud de registro se modifica el estado del usuario a *REFUSED*, indicando que esa solicitud no ha sido aceptada.
- Los botones *Anterior* y *Siguiente* tienen la misma funcionalidad. El primero se muestra si la petición de registro no es la primera, y el segundo si la petición no es la última. Su función es desplazarse en el listado de peticiones, para lo cual modifican el valor del atributo *actualUser* antes de cerrar el componente. Por ello, si en la comprobación de la figura 8.52 este no coincide con *numActualUser*, se vuelve a ejecutar la función *openRequestedDialog()* para que abra la siguiente (o anterior) solicitud de registro de usuario.

En los puntos anteriores se hace referencia al evento *addEvent*, disparado por el componente *table*, como se puede ver en la figura 8.52. Este evento ejecuta la función *add()* del archivo *users.component.ts*; en ella, se utiliza el servicio *users.service.ts* para mandar la consulta HTTP correspondiente al *backend*. Esta consulta es un *POST*, definida en el archivo *auth.js* del *backend*. Su función es tomar el usuario de la solicitud almacenada en la base de datos y cambiar su estado según haya sido aceptado o rechazado, cambiándolo a *ACCEPTED* o *REFUSED*, respectivamente. De esta forma, si el usuario es aceptado, aparecerá a partir de ese momento en el listado de usuarios activos de la web, y se podrá acceder a ella autenticándose con sus credenciales.

Con esta explicación finaliza este capítulo, donde se ha detallado minuciosamente el proceso de implementación de los objetivos propuestos al inicio del TFG. En cada uno de ellos se ha proporcionado una visión completa del desarrollo y las decisiones técnicas que han permitido la creación de la nueva versión de la plataforma, SIMANFOR V2. Además, con esta explicación se establece una base sólida para futuras mejoras y expansiones de la aplicación.

Capítulo 9

Pruebas

Esta sección describe las pruebas que se han llevado a cabo durante el desarrollo del proyecto. Han sido esenciales para verificar que las funcionalidades añadidas a la aplicación son accesibles y operativas para los usuarios de la web, asegurando un correcto funcionamiento del sistema.

La metodología utilizada en estas pruebas ha sido el modelo de caja negra. Este tipo de pruebas se centran en evaluar la funcionalidad del sistema basándose únicamente en los datos de entrada y los resultados obtenidos, sin considerar el conocimiento interno del funcionamiento del sistema. Este enfoque es particularmente útil para identificar discrepancias entre el comportamiento esperado y el real de la aplicación, permitiendo detectar errores que podrían haber sido pasados por alto durante la implementación. Al centrarse en la salida generada por diferentes entradas, se puede garantizar que el sistema responda adecuadamente a las acciones del usuario.

9.1 Pruebas de caja negra

En esta sección, se detallarán los procedimientos y resultados de las pruebas realizadas, exponiéndolas en el mismo orden que se ha utilizado para la explicación de la implementación de los objetivos. Se describirán tanto las pruebas de la API como las interacciones con las interfaces, proporcionando una visión integral de la robustez y fiabilidad del sistema desarrollado. Se muestra la descripción de la prueba, las acciones realizadas, las precondiciones, el valor introducido en el sistema, el resultado esperado y el resultado obtenido, así como el propio resultado final de la prueba. Las pruebas se detallan en las tablas presentadas a continuación, las cuales van desde la 9.1 hasta la 9.28.

PCN - 01	**Entrada de un escenario en la cola de simulaciones**
Descripción	El usuario pulsa el botón *Ejecutar* al abrir los datos de un escenario para realizar su simulación y este entra en la cola del servidor de simulación, cambiando su estado a *WAITING*
Acciones realizadas	1. El usuario pulsa el botón *Detalles* de un escenario en el listado de escenarios 2. El usuario pulsa el botón *Ejecutar* cuando se abre el componente donde se muestran los pasos del escenario
Precondición	El usuario está identificado y el escenario se ha creado
Valor introducido	Ninguno
Resultado esperado	Se cambia el estado del escenario a *WAITING*, y se asigna al campo *jobId* de la base de datos el identificador en la cola del servidor de simulación
Resultado obtenido	Se ha cambiado el estado del escenario a *WAITING* y se ha asignado un valor al campo *jobId* en la base de datos
Resultado de la prueba	Correcto

Tabla 9.1: Prueba de caja negra 1. Entrada de un escenario en la cola de simulaciones

PCN - 02	**Comienzo de la simulación de un escenario**
Descripción	El usuario pulsa el botón *Ejecutar* al abrir los datos de un escenario para realizar su simulación y, tras esperar en la cola, comienza su simulación cambiando su estado a *RUNNING*
Acciones realizadas	1. El usuario pulsa el botón *Detalles* de un escenario en el listado de escenarios 2. El usuario pulsa el botón *Ejecutar* cuando se abre el componente donde se muestran los pasos del escenario 3. Se cambia el estado del escenario a *WAITING* mientras está en cola
Precondición	El usuario está identificado y el escenario se ha creado
Valor introducido	Ninguno
Resultado esperado	Se cambia el estado del escenario a *RUNNING*
Resultado obtenido	Se ha cambiado el estado del escenario a *RUNNING*
Resultado de la prueba	Correcto

Tabla 9.2: Prueba de caja negra 2. Comienzo de la simulación de un escenario

PCN - 03	**Simulación de un escenario**
Descripción	El usuario pulsa el botón *Ejecutar* al abrir los datos de un escenario para realizar su simulación, y cuando termina se cambia su estado a *FINISHED* y se muestra el botón *Resultados*
Acciones realizadas	1. El usuario pulsa el botón *Detalles* de un escenario en el listado de escenarios 2. El usuario pulsa el botón *Ejecutar* cuando se abre el componente donde se muestran los pasos del escenario 3. Se cambia el estado del escenario a *WAITING* mientras está en cola 4. Se cambia el estado del escenario a *RUNNING* mientras se está realizando la simulación
Precondición	El usuario está identificado y el escenario se ha creado
Valor introducido	Ninguno
Resultado esperado	Se cambia el estado del escenario a *FINISHED*, se añade a la base de datos el nombre del archivo de resultados y se muestra el botón *Resultados* en el listado
Resultado obtenido	Se ha cambiado el estado del escenario a *FINISHED*, se ha añadido el nombre del archivo de resultados en la base de datos y se ha mostrado el botón *Resultados* en el listado de escenarios
Resultado de la prueba	Correcto

Tabla 9.3: Prueba de caja negra 3. Simulación de un escenario

PCN - 04	**Mostrar el componente de resultados de un escenario**
Descripción	El usuario pulsa el botón *Resultados* y se abre el componente donde se muestra la información correspondiente
Acciones realizadas	1. El usuario pulsa el botón *Resultados* de un escenario en el listado de escenarios
Precondición	El usuario está identificado y el escenario se ha creado y simulado
Valor introducido	Ninguno
Resultado esperado	Se abre el componente *results* mostrando información del resultado y las opciones de descarga y visualización gráfica
Resultado obtenido	Se ha abierto el componente *results* y se observan la información del resultado y las opciones de descarga y visualización gráfica
Resultado de la prueba	Correcto

Tabla 9.4: Prueba de caja negra 4. Mostrar el componente de resultados de un escenario

PCN - 05	**Descargar el archivo de resultados de un escenario**
Descripción	El usuario pulsa el botón *Descargar archivo* y se descarga un archivo .zip con los resultados de la simulación en una hoja de cálculo Excel
Acciones realizadas	1. El usuario pulsa el botón *Descargar archivo* en el componente de resultados de un escenario
Precondición	El usuario está identificado y se ha abierto el componente *results* de un escenario
Valor introducido	Ninguno
Resultado esperado	Se descarga un archivo .zip que contiene una hoja de cálculo Excel con los resultados de la simulación
Resultado obtenido	Se ha descargado el archivo .zip con la hoja de cálculo Excel de los resultados
Resultado de la prueba	Correcto

Tabla 9.5: Prueba de caja negra 5. Descargar el archivo de resultados de un escenario

PCN - 06	**Abrir la aplicación RStudio Server en el navegador**
Descripción	El usuario pulsa el botón *Ver en RShiny* y se abre en una nueva pestaña del navegador una instancia de RStudio Server con sesión iniciada
Acciones realizadas	1. El usuario pulsa el botón *Ver en RShiny* en el componente de resultados de un escenario
Precondición	El usuario está identificado y se ha abierto el componente *results* de un escenario
Valor introducido	Ninguno
Resultado esperado	Se abre la aplicación RStudio Server en otra pestaña del navegador, conectándose a la última sesión abierta
Resultado obtenido	Se ha abierto RStudio Server con sesión iniciada en otra pestaña del navegador
Resultado de la prueba	Correcto

Tabla 9.6: Prueba de caja negra 6. Abrir la aplicación RStudio Server en el navegador

PCN - 07	**Ver gráficamente los resultados de un escenario antiguo**
Descripción	El usuario abre la conexión con RStudio Server, ejecuta la aplicación Shiny y selecciona su archivo de resultados para verlos gráficamente
Acciones realizadas	1. El usuario escribe *shinyApp(ui, server)* en la consola de RStudio Server 2. El usuario selecciona en el desplegable de la aplicación Shiny el nombre de su archivo de resultados
Precondición	El usuario está identificado y se ha abierto la conexión con RStudio Server
Valor introducido	Función *shinyApp(ui, server)* y nombre del archivo de resultados
Resultado esperado	Se muestran las tablas y gráficos interactivos referidos al archivo de resultados del escenario seleccionado
Resultado obtenido	Se han mostrado las tablas y gráficos interactivos del escenario correspondiente
Resultado de la prueba	Correcto

Tabla 9.7: Prueba de caja negra 7. Ver gráficamente los resultados de un escenario antiguo

PCN - 08	**Ver gráficamente los resultados de un nuevo escenario**
Descripción	El usuario abre la conexión con RStudio Server, reinicia la sesión, ejecuta la aplicación Shiny y selecciona su archivo de resultados para verlos gráficamente
Acciones realizadas	1. El usuario reinicia la sesión de RStudio Server, pulsando el botón rojo de arriba a la derecha y después en *Start New Session* 2. El usuario escribe *shinyApp(ui, server)* en la consola de RStudio Server 3. El usuario selecciona en el desplegable de la aplicación Shiny el nombre de su archivo de resultados
Precondición	El usuario está identificado y se ha abierto la conexión con RStudio Server
Valor introducido	Función *shinyApp(ui, server)* y nombre del archivo de resultados
Resultado esperado	Se muestran las tablas y gráficos interactivos referidos al archivo de resultados del escenario seleccionado
Resultado obtenido	Se han mostrado las tablas y gráficos interactivos del escenario correspondiente
Resultado de la prueba	Correcto

Tabla 9.8: Prueba de caja negra 8. Ver gráficamente los resultados de un nuevo escenario

PCN - 09	**Cierre de sesión**
Descripción	El usuario activo en la web pulsa el nuevo botón y después confirma el cierre de la sesión
Acciones realizadas	1. El usuario pulsa el botón de la parte derecha del menú superior, con el logo de cierre de sesión 2. Se muestra el componente de confirmación con su mensaje correspondiente 3. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario está identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se eliminan el token y el rol del usuario almacenados en *localStorage* y se navega a la página de inicio de la web
Resultado obtenido	Se han eliminado el token y el rol de *localStorage* y se muestra la página de inicio de la web
Resultado de la prueba	Correcto

Tabla 9.9: Prueba de caja negra 9. Cierre de sesión

PCN - 10	**Mostrar listado de usuarios**
Descripción	Un usuario administrador accede a la página *users* y se le muestra el listado de usuarios
Acciones realizadas	1. El usuario administrador pulsa el botón *Gestión de usuarios* del menú lateral
Precondición	El usuario es un administrador identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista de usuarios, mostrando de cada uno su nombre de usuario, nombre, apellidos, centro, departamento, email, teléfono y rol, y la acción de borrar en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado de usuarios con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.10: Prueba de caja negra 10. Mostrar listado de usuarios

PCN - 11	**Mostrar listado de escenarios a un usuario administrador**
Descripción	Un usuario administrador accede a la página *scenarios* y se le muestra el listado con todos los escenarios creados en la web
Acciones realizadas	1. El usuario administrador pulsa el botón *Escenarios* del menú lateral
Precondición	El usuario es un administrador identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista con todos los escenarios de la web, mostrando su ID, autor, inventario utilizado, nombre y clase del modelo utilizado y estado, así como las acciones de acceder a sus datos y borrar (y abrir los resultados en caso de que ya esté simulado) en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado con todos los escenarios de la web, con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.11: Prueba de caja negra 11. Mostrar listado de escenarios a un usuario administrador

PCN - 12	**Mostrar listado de escenarios a un usuario básico**
Descripción	Un usuario básico accede a la página *scenarios* y se le muestra el listado con los escenarios creados por él
Acciones realizadas	1. El usuario básico pulsa el botón *Escenarios* del menú lateral
Precondición	El usuario es un básico identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista con los escenarios creador por el usuario, mostrando su ID, autor, inventario utilizado, nombre y clase del modelo utilizado y estado, así como las acciones de acceder a sus datos y borrar (y abrir los resultados en caso de que ya esté simulado) en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado con los escenarios creados por el usuario básico, mostrando con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.12: Prueba de caja negra 12. Mostrar listado de escenarios a un usuario básico

PCN - 13	**Mostrar listado de inventarios a un usuario administrador**
Descripción	Un usuario administrador accede a la página *inventory* y se le muestra el listado con todos los inventarios de la web
Acciones realizadas	1. El usuario administrador pulsa el botón *Inventarios* del menú lateral
Precondición	El usuario es un administrador identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista con todos los inventarios de la web, tanto los públicos como los privados, mostrando su nombre, fecha de creación, autor, visibilidad y formato, y las acciones de editar y borrar en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado con todos los inventarios de la web, con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.13: Prueba de caja negra 13. Mostrar listado de inventarios a un usuario administrador

PCN - 14	**Mostrar listado de inventarios a un usuario básico**
Descripción	Un usuario básico accede a la página *inventory* y se le muestra el listado con los inventarios subidos por él y los subidos por otros usuarios con visibilidad pública
Acciones realizadas	1. El usuario básico pulsa el botón *Inventarios* del menú lateral
Precondición	El usuario es un básico identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista con los inventarios subidos por el usuario y aquellos de otros usuarios con visibilidad pública, mostrando su nombre, fecha de creación, autor, visibilidad y formato, y las acciones de editar y borrar en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado con los inventarios subidos por el usuario y los subidos por otros usuarios con visibilidad pública, mostrando con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.14: Prueba de caja negra 14. Mostrar listado de inventarios a un usuario básico

PCN - 15	**Mostrar datos de un inventario**
Descripción	Un usuario accede a los datos de un inventario seleccionando el botón de su fila de la tabla correspondiente
Acciones realizadas	1. El usuario pulsa el botón *Editar* de uno de los inventarios de la tabla
Precondición	El usuario está identificado en la web, y en la página *inventory*
Valor introducido	Ninguno
Resultado esperado	Se muestra un formulario con la información de un inventario, con dos tipos de campos: el nombre, el año de creación, la visibilidad y el formato se pueden editar; el ID, el nombre del autor y el nombre del archivo asociado no se pueden editar
Resultado obtenido	Se ha mostrado el formulario con la información del inventario, mostrando los campos indicados
Resultado de la prueba	Correcto

Tabla 9.15: Prueba de caja negra 15. Mostrar datos de un inventario

PCN - 16	**Mostrar listado de modelos**
Descripción	Un usuario accede a la página *models* y se le muestra el listado con todos los modelos de la web
Acciones realizadas	1. El usuario pulsa el botón *Modelos* del menú lateral
Precondición	El usuario está identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista con todos los modelos de la web, mostrando su nombre, tipo, clase, especie, área de aplicación, período de ejecución, cotas de funcionamiento, estado y documentación, y las acciones de editar y borrar en cada uno de ellos
Resultado obtenido	Se ha mostrado el listado con todos los modelos de la web, con los datos indicados
Resultado de la prueba	Correcto

Tabla 9.16: Prueba de caja negra 16. Mostrar listado de modelos

PCN - 17	**Mostrar datos de un modelo**
Descripción	Un usuario accede a los datos de un modelo seleccionando el botón de su fila de la tabla correspondiente
Acciones realizadas	1. El usuario pulsa el botón *Editar* de uno de los modelos de la tabla
Precondición	El usuario está identificado en la web, y en la página *models*
Valor introducido	Ninguno
Resultado esperado	Se muestra un formulario con la información de un modelo, donde se muestra el ID sin poder editarse, y los campos nombre, descripción, documentación, estado, nombre del modelo, clase del modelo, operación, especie, área de aplicación, período de ejecución y cotas de funcionamiento
Resultado obtenido	Se ha mostrado el formulario con la información del modelo, mostrando los campos indicados
Resultado de la prueba	Correcto

Tabla 9.17: Prueba de caja negra 17. Mostrar datos de un modelo

PCN - 18	**Acceso a la documentación de un modelo**
Descripción	Un usuario accede a la documentación de un modelo pulsando el enlace que lleva a la web con la documentación
Acciones realizadas	1. El usuario pulsa palabra *Enlace* de uno de los modelos de la tabla, que contiene el link a la web de la documentación
Precondición	El usuario está identificado en la web, y en la página *models*
Valor introducido	Ninguno
Resultado esperado	Se muestra la página web que contiene la documentación del modelo seleccionado
Resultado obtenido	Se ha mostrado la página web con la documentación del modelo
Resultado de la prueba	Correcto

Tabla 9.18: Prueba de caja negra 18. Acceso a la documentación de un modelo

PCN - 19	**Acceso a la página de inicio de sesión**
Descripción	Un usuario accede a la página de inicio de sesión al pulsar el botón correspondiente en la página principal
Acciones realizadas	1. El usuario pulsa el botón *Iniciar Sesión* disponible en las páginas de la web cuando este no está autenticado
Precondición	Ninguna
Valor introducido	Ninguno
Resultado esperado	Se muestra la página web de inicio de sesión
Resultado obtenido	Se ha mostrado la página web de inicio de sesión
Resultado de la prueba	Correcto

Tabla 9.19: Prueba de caja negra 19. Acceso a la página de inicio de sesión

PCN - 20	**Añadir solicitud de registro de usuario**
Descripción	Un usuario rellena el formulario de solicitud de registro correctamente, y se crea el usuario en la base de datos
Acciones realizadas	1. El usuario pulsa el botón *Solicitud de Registro* de la página de inicio de sesión 2. Se muestra el componente con el formulario de solicitud 3. El usuario rellena los campos del formulario, de forma que las contraseñas coincidan en ambos campos 4. El usuario pulsa el botón *Aceptar* del formulario 5. Se muestra el componente de confirmación con su mensaje correspondiente 6. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario está en la página de inicio de sesión
Valor introducido	Se escriben en el formulario de solicitud: nombre de usuario, contraseña (igual en los dos campos), nombre, apellidos, centro, departamento, email, teléfono y rol
Resultado esperado	Se añade el usuario a la base de datos con atributo *status* como *REQUESTED*
Resultado obtenido	Se ha añadido el usuario correctamente a la base de datos
Resultado de la prueba	Correcto

Tabla 9.20: Prueba de caja negra 20. Añadir solicitud de registro de usuario

PCN - 21	**Añadir solicitud de registro de usuario con contraseñas distintas**
Descripción	Un usuario rellena el formulario de solicitud de registro, sin igualar los valores de los dos campos de las contraseñas
Acciones realizadas	1. El usuario pulsa el botón *Solicitud de Registro* de la página de inicio de sesión 2. Se muestra el componente con el formulario de solicitud 3. El usuario rellena los campos del formulario, sin igualar los valores de los campos de las contraseñas
Precondición	El usuario está en la página de inicio de sesión
Valor introducido	Se escriben en el formulario de solicitud: nombre de usuario, contraseña (distinta en los dos campos), nombre, apellidos, centro, departamento, email, teléfono y rol
Resultado esperado	Se muestra el mensaje de error correspondiente y no se deja enviar la solicitud
Resultado obtenido	Se ha mostrado el mensaje de error y el botón *Aceptar* no está disponible, impidiendo enviar la solicitud
Resultado de la prueba	Correcto

Tabla 9.21: Prueba de caja negra 21. Añadir solicitud de registro de usuario con contraseñas distintas

PCN - 22	**Añadir solicitud de registro de usuario con nombre de usuario existente**
Descripción	Un usuario rellena el formulario de solicitud de registro, utilizando un nombre de usuario ya existente en la web
Acciones realizadas	1. El usuario pulsa el botón *Solicitud de Registro* de la página de inicio de sesión 2. Se muestra el componente con el formulario de solicitud 3. El usuario rellena los campos del formulario, indicando en el nombre de usuario uno existente en la web 4. El usuario pulsa el botón *Aceptar* del formulario 5. Se muestra el componente de confirmación con su mensaje correspondiente 6. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario está en la página de inicio de sesión, y ya existe un usuario con ese nombre de usuario en el sistema
Valor introducido	Se escriben en el formulario de solicitud: nombre de usuario, contraseña (igual en los dos campos), nombre, apellidos, centro, departamento, email, teléfono y rol
Resultado esperado	Se detecta que ya existe un usuario con ese nombre de usuario, se rechaza la solicitud de registro y se muestra el mensaje de error correspondiente
Resultado obtenido	Se ha rechazado la petición y se ha mostrado el mensaje con la información del error
Resultado de la prueba	Correcto

Tabla 9.22: Prueba de caja negra 22. Añadir solicitud de registro de usuario con nombre de usuario existente

PCN - 23	**Añadir solicitud de registro de usuario con email existente**
Descripción	Un usuario rellena el formulario de solicitud de registro, utilizando un email ya existente en la web
Acciones realizadas	1. El usuario pulsa el botón *Solicitud de Registro* de la página de inicio de sesión 2. Se muestra el componente con el formulario de solicitud 3. El usuario rellena los campos del formulario, indicando en el email uno existente en la web 4. El usuario pulsa el botón *Aceptar* del formulario 5. Se muestra el componente de confirmación con su mensaje correspondiente 6. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario está en la página de inicio de sesión, y ya existe un usuario con ese email en el sistema
Valor introducido	Se escriben en el formulario de solicitud: nombre de usuario, contraseña (igual en los dos campos), nombre, apellidos, centro, departamento, email, teléfono y rol
Resultado esperado	Se detecta que ya existe un usuario con ese email, se rechaza la solicitud de registro y se muestra el mensaje de error correspondiente
Resultado obtenido	Se ha rechazado la petición y se ha mostrado el mensaje con la información del error
Resultado de la prueba	Correcto

Tabla 9.23: Prueba de caja negra 23. Añadir solicitud de registro de usuario con email existente

PCN - 24	**Acceso a la nueva página de usuarios**
Descripción	Un usuario administrador accede a la página *users* y se le muestra el listado de usuarios activos en el sistema, junto con el botón que indica las solicitudes pendientes
Acciones realizadas	1. El usuario administrador pulsa el botón *Gestión de usuarios* del menú lateral
Precondición	El usuario es un administrador identificado en la web
Valor introducido	Ninguno
Resultado esperado	Se muestra la lista de usuarios que tienen el atributo *status* distinto de *REQUESTED* y *REFUSED*, es decir, los usuarios activos; también se muestra el botón que abre las solicitudes de registro, donde se indica cuántas solicitudes quedan pendientes
Resultado obtenido	Se ha mostrado el listado de usuarios activos, junto con el botón para abrir las solicitudes de registro y el número de estas que quedan pendientes
Resultado de la prueba	Correcto

Tabla 9.24: Prueba de caja negra 24. Acceso a la nueva página de usuarios

PCN - 25	**Aceptar solicitud de registro de usuario**
Descripción	Un usuario administrador accede a una solicitud de registro de nuevo usuario, y pulsa el botón para aceptar la petición
Acciones realizadas	1. El usuario administrador pulsa el botón *Solicitudes de Nuevo Usuario* de la página *users* 2. Se muestra un formulario con los datos de la primera petición pendiente, sin posibilidad de editar estos datos 3. El usuario pulsa el botón *Aceptar* para aceptar la solicitud 4. Se muestra el componente de confirmación con su mensaje correspondiente 5. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario es un administrador identificado en la web y en la página *users*
Valor introducido	Ninguno
Resultado esperado	Se cambia la variable *status* del usuario a *ACCEPTED* en la base de datos, apareciendo a partir de ese momento en el listado de usuario activos; también se decrementa en uno el número de solicitudes pendientes del botón *Solicitudes de Nuevo Usuario* de la página *users*
Resultado obtenido	Se ha cambiado la variable *status* del usuario a *ACCEPTED* y se ve el usuario en el listado de activos; se ha decrementado en uno el número de solicitudes pendientes
Resultado de la prueba	Correcto

Tabla 9.25: Prueba de caja negra 25. Aceptar solicitud de registro de usuario

PCN - 26	**Rechazar solicitud de registro de usuario**
Descripción	Un usuario administrador accede a una solicitud de registro de nuevo usuario, y pulsa el botón para rechazar la petición
Acciones realizadas	1. El usuario administrador pulsa el botón *Solicitudes de Nuevo Usuario* de la página *users* 2. Se muestra un formulario con los datos de la primera petición pendiente, sin posibilidad de editar estos datos 3. El usuario pulsa el botón *Rechazar* para rechazar la solicitud 4. Se muestra el componente de confirmación con su mensaje correspondiente 5. El usuario pulsa el botón *Si* del componente de confirmación
Precondición	El usuario es un administrador identificado en la web y en la página *users*
Valor introducido	Ninguno
Resultado esperado	Se cambia la variable *status* del usuario a *REFUSED* en la base de datos; también se decrementa en uno el número de solicitudes pendientes del botón *Solicitudes de Nuevo Usuario* de la página *users*
Resultado obtenido	Se ha cambiado la variable *status* del usuario a *REFUSED* y se ha decrementado en uno el número de solicitudes pendientes
Resultado de la prueba	Correcto

Tabla 9.26: Prueba de caja negra 26. Rechazar solicitud de registro de usuario

PCN - 27	**Mostrar la siguiente petición de registro de usuario**
Descripción	Un usuario administrador accede a una solicitud de registro de nuevo usuario, y pulsa el botón para mostrar la siguiente petición pendiente
Acciones realizadas	1. El usuario administrador pulsa el botón *Solicitudes de Nuevo Usuario* de la página *users* 2. Se muestra un formulario con los datos de la primera petición pendiente, sin posibilidad de editar estos datos 3. El usuario pulsa el botón *Siguiente* para acceder a la siguiente solicitud
Precondición	El usuario es un administrador identificado en la web y en la página *users*; existen dos o más solicitudes pendientes de registro
Valor introducido	Ninguno
Resultado esperado	Se muestra la siguiente petición de registro de usuario del listado de pendientes, abriendo el formulario con los datos de la solicitud, y pudiendo realizar la acciones de aceptar y rechazar la petición
Resultado obtenido	Se ha mostrado el formulario con los datos de la siguiente solicitud, y se pueden realizar la acciones de aceptar y rechazar sobre ella
Resultado de la prueba	Correcto

Tabla 9.27: Prueba de caja negra 27. Mostrar la siguiente petición de registro de usuario

PCN - 28	**Mostrar la anterior petición de registro de usuario**
Descripción	Un usuario administrador accede a una solicitud de registro de nuevo usuario, y pulsa el botón para mostrar la anterior petición pendiente
Acciones realizadas	1. El usuario administrador pulsa el botón *Solicitudes de Nuevo Usuario* de la página *users* 2. Se muestra un formulario con los datos de la primera petición pendiente, sin posibilidad de editar estos datos 3. El usuario pulsa el botón *Siguiente* para acceder a la siguiente solicitud 4. El usuario pulsa el botón *Anterior* para acceder a la anterior solicitud
Precondición	El usuario es un administrador identificado en la web y en la página *users*; existen dos o más solicitudes pendientes de registro
Valor introducido	Ninguno
Resultado esperado	Se muestra la anterior petición de registro de usuario del listado de pendientes, abriendo el formulario con los datos de la solicitud, y pudiendo realizar la acciones de aceptar y rechazar la petición
Resultado obtenido	Se ha mostrado el formulario con los datos de la anterior solicitud, y se pueden realizar la acciones de aceptar y rechazar sobre ella
Resultado de la prueba	Correcto

Tabla 9.28: Prueba de caja negra 28. Mostrar la anterior petición de registro de usuario

Capítulo 10

Conclusiones

El proyecto concluye con una nueva versión completamente funcional y escalable de la web SIMANFOR, denominada SIMANFOR V2. A partir del desarrollo de este proyecto, se pueden extraer las siguientes conclusiones:

- Se ha seguido un proceso similar al que realizan las empresas contratadas por clientes, empezando con el análisis del código y del sistema existente antes de realizar las modificaciones de la implementación de objetivos
- El desarrollo del proyecto se ha llevado a cabo en una máquina virtual. Una vez configurado el entorno de trabajo, esto facilitó significativamente el proceso de desarrollo.
- Se ha utilizado el ecosistema Docker de la web original, incorporando mejoras para optimizar la aplicación.
- Se logró el objetivo principal: integrar la aplicación RShiny en el ecosistema de la web SIMANFOR.
- Se alcanzaron otros objetivos que mejoraron la experiencia del usuario, resultando en una versión de la aplicación más amigable y fácil de usar.
- El proceso de desarrollo se realizó de forma incremental, permitiendo un control exhaustivo del progreso del proyecto.
- Todos los requisitos definidos en la fase de análisis fueron cumplidos y verificados mediante una fase final de pruebas.

10.1 Experiencia personal

La mayor contribución a mi experiencia personal como ingeniero de software ha sido participar en el proceso de ser contactado por una empresa cliente y desarrollar un software para ellos. En particular, destaco la primera

fase del proyecto, en la que realicé la comprensión y documentación de la aplicación existente, lo cual facilitó enormemente el desarrollo posterior.

Además, considero muy enriquecedor haber trabajado con una estructura de micro-servicios como Docker. Este tipo de software es ampliamente utilizado en la actualidad, y familiarizarme con él me prepara mejor para futuros proyectos. El uso de tecnologías como Angular para el frontend y Node.js para el *backend* también ha ampliado mi conocimiento sobre las herramientas utilizadas en aplicaciones web reales.

Planificar el proyecto con un enfoque de desarrollo incremental y dividirlo en objetivos me permitió mantener un control preciso del progreso, facilitando la gestión de las tareas. A pesar de los retrasos en algunas actividades, como la configuración del entorno de desarrollo, una buena planificación inicial ayudó a compensar estos inconvenientes.

Por último, trabajar con SIMANFOR me permitió entender la estructura y las herramientas utilizadas en una web real lanzada al mercado. Esto incluyó la configuración de un ecosistema Docker con servicios de base de datos, *backend*, frontend y un proxy para el manejo de peticiones.

10.2 Trabajo futuro

Tras haber alcanzado los objetivos propuestos, se sugieren las siguientes tareas para continuar con la expansión de la funcionalidad del sistema. Estas están relacionadas con las nuevas características de SIMANFOR V2 y no consideran otras modificaciones de la web original:

- Según se ha realizado la conexión con la aplicación RShiny, solo puede haber una sesión activa de RStudio Server. Esto implica que cuando un nuevo usuario se conecta al servicio del contenedor Docker, la sesión anterior se cierra, sin permitir el trabajo de dos o más sesiones en paralelo. Se sugiere permitir conexiones simultáneas de múltiples usuarios al servicio de RStudio Server para facilitar trabajos distintos en paralelo.

- Para cargar un nuevo escenario en la aplicación RShiny, es necesario reiniciar la sesión de RStudio Server. De esta forma, el archivo *Shiny_completo.R* realiza los cálculos sobre sus variables y lo deja disponible para su visualización en la aplicación Shiny. Se propone eliminar esta necesidad, permitiendo la carga de nuevos escenarios al abrir la conexión con el servicio.

- El archivo *Shiny_completo.R* carga todos los escenarios disponibles en el directorio *data* del contenedor Docker de RStudio Server. Se sugiere modificarlo para que solo cargue un archivo de resultados específico cuando se ejecute. Durante el desarrollo del proyecto se buscaron varias formas de hacerlo, aunque no se consiguió con ninguna de ellas: a través de la configuración de volúmenes Docker, utilizando la URL con la que se abre RStudio Server o usando el servicio de API del propio RStudio Server.

- El archivo *Shiny_completo.R* proporcionado por SIMANFOR tenía soporte limitado para hojas de cálculo en español. Aunque en este proyecto se consiguió dar soporte para cargar más de una hoja de cálculo, no se consiguió corregir el tratamiento de las variables relacionadas con el volumen de las parcelas. Se recomienda revisar esta funcionalidad para asegurar la correcta carga de todas las variables.

- Se implementó un enlace a la documentación de cada modelo listado en su tabla correspondiente. Se sugiere revisar estos enlaces para asegurar que abren correctamente la página de destino.

- La versión inicial de la web requería descargar y enviar un PDF para la solicitud de nuevas cuentas de usuario. Se propone la posibilidad de añadir nueva información al formulario de solicitud de registro, incluyendo nuevos campos para rellenar (de tipo *string*) o mostrar información (de tipo *show*) según sea necesario. Para ello, estos campos se deberían añadir al atributo *addForm* del archivo *login.component.ts*.

Appendices

Apéndice A

Manual de Instalación

En este manual de instalación se describen los pasos necesarios para desplegar la aplicación web SIMANFOR en un entorno local. Esto se debe a que el proyecto no ha sido realizado en su entorno de producción, si no en una máquina virtual que proporciona una plataforma de desarrollo y prueba para trabajar con el proyecto de manera totalmente libre y eficiente.

Para el desarrollo en local, se ha utilizado una de las máquinas virtuales del sistema de virtualización utilizado por la Escuela de Ingeniería Informática de la UVa. Se trata de una Ubuntu Server 22.04, con 4 GB de memoria y 32 GB de disco. Su número es el 2003X, utilizando el último dígito como método para indicar el modo de conexión desde el ordenador personal: 1 para SSH, 2 para HTTP y 3 para HTTPS. Por tanto, el acceso a ella se realiza a través de Visual Studio Code, conectándose a la dirección *virtual.lab.inf.uva.es*, a través del puerto 20031, y utilizando el usuario *rafagc* junto con una clave privada que fue proporcionada junto a la máquina.

Por otro lado, la máquina virtual cuenta con la herramienta Docker instalada previamente, dando soporte a Docker Compose y Docker Buildx. Sin embargo, en caso de trabajar en una máquina Linux sin este software incluido en ella, se proporcionan los siguientes pasos para llevar a cabo la instalación a través de su consola de comandos:

1. Actualizar el sistema:

```
$ sudo apt update
$ sudo apt upgrade
```

2. Instalar Docker:

```
$ sudo apt install docker.io
```

3. Iniciar y habilitar Docker

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

4. Verificar la instalación de Docker:

```
$ docker --version
```

5. Instalar Docker Compose:

```
$ sudo apt install docker-compose
```

6. Verificar la instalación de Docker Compose:

```
$ docker-compose --version
```

Tras instalar Docker, se debe utilizar Git para obtener el código fuente del proyecto. Este se encuentra en un repositorio de Github, que contiene cuatro directorios correspondientes a los repositorios originales entregados por SIMANFOR. Los pasos para instalar Git y obtener el código fuente son los siguientes:

1. Instalar Git:

```
$ sudo apt install git
```

2. Verificar la instalación de Git:

```
$ git --version
```

3. Clonar el siguiente repositorio desde Github:

```
$ git clone https://github.com/rafgome/TFG_SimanforV2.git
```

El siguiente paso consiste en la puesta en marcha de los servicios Docker. Para ello se utiliza el archivo *docker-compose.yml*, por lo que es necesario navegar previamente al directorio donde está almacenado (en este caso, está en el directorio *production*). Se deben modificar las rutas de los volúmenes de los contenedores del *backend*, base de datos y RStudio Server, así como las contraseñas según se indica en el archivo README.md. Posteriormente, se cargan las imágenes de Docker y se inician los servicios para poner en marcha la aplicación:

1. Navegar al directorio del repositorio *production*, donde se encuentra el archivo *docker-compose.yml*:

```
$ cd production
```

2. Construir las imágenes de Docker:

   ```
   $ docker compose build
   ```

3. Iniciar los servicios:

   ```
   $ docker compose up
   ```

4. Verificar que los servicios están corriendo:

   ```
   $ docker compose ps
   ```

5. Detener los servicios:

   ```
   $ docker compose down
   ```

Con esto, la aplicación Docker estaría funcionando, con todos los servicios corriendo y escuchando en sus puertos correspondientes.

Apéndice B

Manual de Usuario

Una vez que el usuario ha conseguido el acceso al entorno local de la plataforma, se presenta una gran variedad de funcionalidades a realizar. La mayoría de estas están detalladas en el *Manual de Usuario* creado por el equipo de SIMANFOR [48]. Sin embargo, a continuación se va a detallar la forma en la que el usuario puede acceder a las nuevas funcionalidades añadidas en este proyecto.

En primer lugar, para ver la funcionalidad relacionada con RShiny, se debe crear un escenario, utilizando un inventario y un modelo de proyección adecuados. Una vez ejecutada la simulación, se debe pulsar el botón *Resultados* que abre la vista con los resultados del escenario. En ella, al pulsar el botón *Ver en RShiny* se abre en una nueva pestaña una instancia de RStudio Server.

En la nueva pestaña, el usuario debe reiniciar la sesión de RStudio Server, utilizando el botón rojo de arriba a la derecha y pulsando después en *Start New Session*. Cuando se reinicia la sesión, se debe escribir en la consola de comandos la función *shinyApp(ui, server)*, de forma que se abra la aplicación Shiny en una nueva pestaña. En la aplicación Shiny se puede seleccionar el archivo de resultados cuyo nombre coincide con el escenario simulado, permitiendo así ver los resultados de la simulación de una manera gráfica e interactiva.

La siguiente funcionalidad añadida corresponde con el botón de cierre de sesión. En la nueva versión, se muestra dicho botón en la parte superior derecha de las páginas de la web. El usuario simplemente pulsa el botón para cerrar su sesión actual, aceptando en la vista de confirmación que se muestra a continuación.

Otra nueva característica de la web consiste en cómo se reorganizan las tablas con los listados de usuarios, inventarios, escenarios y modelos. Al acceder a cada una de las páginas mencionadas (a la página de usuarios solo pueden acceder los administradores), se observa cómo la información que se muestra es mucho más clara y concisa. Además, al pulsar el botón para editar un elemento, el usuario puede observar nuevos campos en los que se muestra información adicional del elemento a editar.

En relación con lo anterior, cuando un usuario accede al listado de modelos se observa una columna que contiene los enlaces a la documentación de cada uno de los modelos disponibles. Esta nueva funcionalidad permite al usuario pulsar en sobre la palabra clave *Enlace* para acceder a la dirección web donde se encuentra

dicha documentación.

La última funcionalidad añadida corresponde al registro de usuarios en la web. Para ello, si un usuario quiere solicitar la creación de una nueva cuenta, deberá rellenar el formulario que se abre al pulsar el botón *Solicitud de Registro* en la página de inicio de sesión. Una vez rellenado correctamente, se crea la solicitud para que el equipo de SIMANFOR la pueda validar.

Desde el punto de vista de un usuario administrador, cuando este accede a la página con el listado de usuarios se muestra un nuevo botón, *Solicitudes de Nuevo Usuario*. En él también se indican cuántas solicitudes hay pendientes en ese momento, permitiendo al administrador pulsar el botón que abre la información detallada de la primera de las solicitudes. El administrador puede navegar entre las distintas solicitudes, y confirmar o rechazar cada una de ellas, creando o eliminando un nuevo usuario en la web.

Apéndice C

Manual del Desarrollador

Desde el punto de vista del desarrollador, el estado del proyecto es completamente funcional, con pruebas realizadas con éxito sobre las nuevas funcionalidades añadidas. En caso de querer ver cómo se ha estructurado y realizado la implementación del proyecto, se pueden ver los capítulos 6, 7 y 8.

Por otro lado, en la sección 10.2 se describen nuevas tareas e incrementos que se pueden realizar a partir de la versión de SIMANFOR desarrollada en este proyecto. Estos nuevos objetivos están relacionados solamente con la nueva funcionalidad incluida durante el desarrollo de este TFG, sin incluir otras áreas de la aplicación.

En caso de trabajar con el entorno local de desarrollo descrito en este proyecto, se describen varias acciones a realizar para limpiar periódicamente la estructura Docker sobre la que se despliega la aplicación:

- Para limpiar los contenedores e imágenes sueltas, es decir, que están creadas pero no están siendo utilizadas, escribir el siguiente comando:

  ```
  $ docker system prune
  ```

- En caso de que no se hayan eliminado todas las imágenes con el uso del comando anterior, se puede utilizar el siguiente comando para forzar la limpieza de dichas imágenes:

  ```
  $ docker image prune -a
  ```

- Para comprobar el espacio utilizado por Docker, escribir el siguiente comando, donde se describen los servicios y su espacio ocupado:

  ```
  $ docker ps
  ```

Los comandos anteriores aseguran que la estructura Docker no agota el almacenamiento de la máquina

utilizada para el desarrollo del proyecto. Para desplegar la aplicación, se deben seguir los pasos relacionados con la estructura Docker descritos en el apéndice A.

Con estas instrucciones se asegura que un desarrollador de software sea capaz de entender la aplicación implementada, desplegar el ecosistema Docker sobre el que se sustenta y establecer un punto de partida para realizar trabajos futuros sobre la plataforma.

Bibliografía

1. Pretzsch, H. *Forest Dynamics, Growth and Yield* (Springer, 2009).
2. Bravo, F. y Ordoñez, C. *SIMANFOR: Sistema de apoyo para la simulación de alternativas de manejo forestal sostenible, 2021* https://www.simanfor.es/ (2024).
3. *SNGULAR* https://www.sngular.com/ (2024).
4. Pérez, A. Características y fases del modelo incremental. *OBS Business School*. https://www.obsbusiness.school/blog/caracteristicas-y-fases-del-modelo-incremental (2016).
5. *Working with the Container registry* https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry (2023).
6. Duncan, A. How Can We Easily and Visually Explain Docker-Compose? *Medium*. https://medium.com/clarusway/how-can-we-easily-and-visually-explain-the-docker-compose-53df77e9f046 (2020).
7. *Docker Docs. Volumes* https://docs.docker.com/storage/volumes/ (2023).
8. *Nginx* https://www.nginx.com/ (2024).
9. Ellingwood, J. Understanding the Nginx Configuration File Structure and Configuration Contexts. *DigitalOcean*. https://www.digitalocean.com/community/tutorials/understanding-the-nginx-configuration-file-structure-and-configuration-contexts (2022).
10. *Swagger UI* https://swagger.io/tools/swagger-ui/ (2023).
11. Ellis, D. R. What is Swagger? A Beginner's Guide. *HubSpot*. https://blog.hubspot.com/website/what-is-swagger#:~:text=Swagger%20API%20is%20a%20set,known%20as%20the%20OpenAPI%20specification (2022).
12. *nodeJs* https://nodejs.org/en (2024).
13. *Angular* https://angular.io/ (2024).
14. *Karma* https://karma-runner.github.io/latest/index.html (2023).
15. *Protractor* https://www.protractortest.org/#/ (2023).
16. *TSConfig Reference* https://www.typescriptlang.org/tsconfig (2023).
17. *Angular workspace configuration* https://angular.io/guide/workspace-config (2023).
18. *Firebase Hosting* https://firebase.google.com/docs/hosting?hl=es (2023).
19. *Microsoft Teams* https://www.microsoft.com/es-es/microsoft-teams/group-chat-software/ (2024).
20. *Atlassian Trello* https://trello.com/es (2024).
21. *Figma* https://www.figma.com/ (2024).

22. *Github* https://github.com/ (2024).
23. *Astah Professional* https://astah.net/products/astah-professional/ (2024).
24. *Microsoft Word* https://www.microsoft.com/es-es/microsoft-365/word (2024).
25. *Overleaf* https://es.overleaf.com (2024).
26. *LaTeX* https://www.latex-project.org/ (2024).
27. *MobaXterm* https://mobaxterm.mobatek.net/ (2024).
28. *Visual Studio Code* https://code.visualstudio.com/ (2024).
29. *Ubuntu* https://ubuntu.com/ (2024).
30. *Docker* https://www.docker.com (2024).
31. *MongoDB* https://www.mongodb.com/ (2024).
32. *Javascript* https://www.javascript.com/ (2024).
33. *Typescript* https://www.typescriptlang.org/ (2024).
34. *HTML* https://html.com/ (2024).
35. *CSS* https://www.w3.org/Style/CSS/Overview.en.html (2024).
36. *Rocker* https://rocker-project.org/ (2024).
37. *rocker/verse* https://hub.docker.com/r/rocker/verse (2024).
38. *R* https://www.r-project.org/ (2024).
39. *Shiny* https://shiny.posit.co/ (2024).
40. *ChatGPT* https://chat.openai.com/ (2024).
41. *Microsoft Excel* https://www.microsoft.com/es-es/microsoft-365/excel (2024).
42. Lewis, J. y Fowler, M. Microservices. *martinFowler.com*. https://martinfowler.com/articles/microservices.html (2014).
43. Stonis, M. y Pine, D. Modelo-Vista-Modelo de vista (MVVM). *Microsoft Build*. https://learn.microsoft.com/es-es/dotnet/architecture/maui/mvvm (2023).
44. Schmitt, J. Git tags vs branches: Differences and when to use them. *CircleCI Blog*. https://circleci.com/blog/git-tags-vs-branches/ (2023).
45. *.Rprofile* https://docs.posit.co/ide/user/ide/guide/environments/r/managing-r.html#rprofile (2024).
46. *Youtube* https://www.youtube.com/ (2024).
47. *LinkedIn* https://www.linkedin.com/ (2024).
48. *Manual de usuario de SIMANFOR* https://www.simanfor.es/assets/documents/SIMANFOR_manual.pdf (2023).